

Trimesteroverschrijdend Project: Curve Editor

Groep 13: Sibrand Staessens en Sibren Polders

Donderdag 22 mei, 2008

Inhoudsopgave

1 Voorwoord	3
2 Wiskundige voorkennis	4
2.1 Lineaire interpolatie	4
2.1.1 Naïeve poging (brute force)	4
2.1.2 DDA-interpolatie (Digital Differential Analyzer)	4
2.2 Bézier	4
2.2.1 Naïeve poging (brute force)	4
2.2.2 Forward differences	5
2.2.3 C0-continuïteit	6
2.2.4 G1-continuïteit	6
2.2.5 C1-continuïteit	7
2.3 Hermite	8
3 Implementatie	9
3.1 Packages	9
3.1.1 Java packages	9
3.1.2 CurveEditor	9
3.1.3 Algorithms	9
3.1.4 Core	10
3.1.5 Curves	11
3.1.6 Exceptions	11
3.1.7 GUI	12
3.1.8 Tools	12
3.2 Uitwerking van de GUI	12
3.2.1 Overzicht	12
3.3 Listeners in Java	13
3.4 Datastructuren	14
3.4.1 Point	14
3.4.2 Curve	14
3.5 Extra's	15
3.5.1 Path Simulation Tool	15
3.6 Interessante problemen	15
3.6.1 Vector vs. LinkedList	15
3.6.2 HashMap vs. 2D Matrix	15
3.6.3 Systeemafhankelijke problemen	17
4 Planning	18
5 Takkverdeling	27
6 Appendix	28

A Screenshots	28
B UML-diagrammen	35
C Referenties	40
D Handleiding	40

1 Voorwoord

Op welke manieren kan je een set van punten op een vloeiende wijze met elkaar verbinden en hoe ziet het bekomen pad er dan uit ? Dat is de vraag waarvoor Curve Editor een gedeeltelijke oplossing biedt. Het meest voor de hand liggende pad dat tussen de punten getekend kan worden is uiteraard dat pad dat bekomen wordt m.b.v. lineaire interpolatie. Maar er zijn nog zoveel andere mogelijkheden, waarvan er in Curve Editor twee verwerkt zijn (nl. Bzier- en Hermite-curves). Deze algoritmes zullen in vgl. met lineaire interpolatie vloeiende krommen tussen de interpolatiepunten creëren. De toepassingen van de door Curve Editor gebruikte algoritmes beperken zich niet enkel tot het tekenen van "lijntjes" tussen punten, maar worden bijvoorbeeld ook veelvuldig gebruikt voor vloeiende camerabewegingen of "AI"-bewegingen in games. De interpolatie van een gegeven set van controlepunten is een uitgebreide en interessante studie die op vele vlakken in de informatica/wiskunde zijn nut kan bewijzen. Curve Editor geeft er de basistoepassing van, namelijk die van de wiskundige berekening en grafische voorstelling van de gevraagde curve.

"If the path be beautiful, let us not ask where it leads." - Anatole France

2 Wiskundige voorkennis

2.1 Lineaire interpolatie

(zie figuur 7)

2.1.1 Naïeve poging (brute force)

Een eerste poging om een lineaire interpolatie te berekenen is die waarbij tussen elk paar van controlepunten de vergelijking $y = mx + b$ wordt uitgerekend. Hierbij geldt: $m = (y_1 - y_0)/(x_1 - x_0)$, en b is het startpunt op de y-as. De grote nadelen van dit algoritme zijn dat er voor iedere nieuwe pixel opnieuw berekend en getekend moet worden en dat er behoeft is aan een floating-point optelling en vermenigvuldiging (vermits m een floating-point getal is).

2.1.2 DDA-interpolatie (Digital Differential Analyzer)

Bij deze methode wordt nog steeds m als een floating-point getal voorgesteld, maar de berekening van de volgende y-waarde wordt als volgt vereenvoudigd:

$$\begin{aligned}y_{i+1} &= m \cdot x_{i+1} + b \\&= m \cdot (x_i + \Delta x) + b \\&= m \cdot x_i + m \cdot \Delta x + b \\&= y_i + m \cdot \Delta x\end{aligned}$$

= $y_i + m$ aangezien $\Delta x = 1$ als we 1 pixel verder gaan

Dit algoritme is duidelijk efficiënter vermits we een floating-point vermenigvuldiging hebben verwijderd.

2.2 Bézier

Het cubic Bzier-algoritme construeert een curve als volgt: per vier opeenvolgende controlepunten wordt een deel van de totale curve berekend. Die deelcurve start in het eerste controlepunt, in de richting van het tweede, om dan in het vierde controlepunt te eindigen in de richting van het derde. Een volgende deelcurve heeft als eerste controlepunt het laatste controlepunt van de vorige deelcurve om also C0-continuïteit te garanderen.

2.2.1 Naïeve poging (brute force)

Bij een eerste poging kunnen we het algoritme implementeren a.h.v. de meetkundige constructie en de parametrische voorstelling van de curve. Deze is als volgt:

$$\mathbf{B}(t) = (1-t)^3 \mathbf{P}_0 + 3t(1-t)^2 \mathbf{P}_1 + 3t^2(1-t) \mathbf{P}_2 + t^3 \mathbf{P}_3, \quad t \in [0, 1].$$

Indien we nu voor een eindig aantal t-waarden tussen 0 en 1, de B-waarde berekenen, dan hebben we een eindig aantal punten berekend die op de gevraagde curve liggen. Gebruiken we nadien lineaire interpolatie om de berekende punten met elkaar te verbinden, dan hebben een eerste benadering van de cubic Bézier-curve. Het spreekt voor zich dat deze implementatie verre van efficiënt is: voor elke t-waarde moeten we de machten uitrekenen, vermenigvuldigen met de coördinaten van de controlepunten en daarna alles optellen. Dit is tijdsintensief rekenwerk, daarom is in Curve Editor het incrementeel algoritme, dat in de volgende sectie besproken wordt, geïmplementeerd.

2.2.2 Forward differences

Deze implementatie is een incrementeel algoritme: de volgende berekening wordt gedaan a.h.v. wat in de vorige berekening werd gevonden. Op deze manier moet men voor elk gevraagd punt niet helemaal van nul beginnen en minder rekenwerk uitvoeren. Hoe we tot de implementatie komen, gebeurt als volgt:

We evalueren de polynoom in een aantal even ver van elkaar liggende punten. Stel dat we $f(t) = at^3 + bt^2 + ct + d$ willen evalueren voor de punten t_0, t_1, \dots, t_m met $t_{i+1} = t_i + \epsilon$ voor $i = 0, 1, \dots, m - 1$. Dan bekomen we:

$$\begin{aligned} f_{i+1} &= a(t_i + \epsilon)^3 + b(t_i + \epsilon)^2 + c(t_i + \epsilon) + d \\ &= f_i + (3a\epsilon)t_i^2 + (3a\epsilon^2 + 2b\epsilon)t_i + (a\epsilon^3 + b\epsilon^2 + c\epsilon) \\ &= f_i + \Delta f_i \end{aligned}$$

We noemen Δf_i de voorwaartse differentie. Mer op dat Δf_i een 2e-graadspolynoom is, een graad lager dan de originele polynoom. We passen hetzelfde toe voor Δf_i :

$$\begin{aligned} \Delta f_{i+1} &= (3a\epsilon)(t_i + \epsilon)^2 + (3a\epsilon^2 + 2b\epsilon)(t_i + \epsilon) + (a\epsilon^3 + b\epsilon^2 + c\epsilon) \\ &= \Delta f_i + (6a\epsilon^2)t_i + (6a\epsilon^3 + 2b\epsilon^2) \\ &= \Delta f_i + \Delta^2 f_i \end{aligned}$$

En voor $\Delta^2 f_i$:

$$\begin{aligned} \Delta^2 f_{i+1} &= (6a\epsilon^2)(t_i + \epsilon) + (6a\epsilon^3 + 2b\epsilon^2) \\ &= \Delta^2 f_i + 6a\epsilon^3 = \Delta^2 f_i + \Delta^3 f \end{aligned}$$

Samengevat hebben we dus:

$$\begin{aligned}
f_i &= at_i^3 + bt_i^2 + ct_i + d \\
\Delta f_i &= (3a\epsilon)t_i^2 + (3a\epsilon^2 + 2b\epsilon)t_i + (a\epsilon^3 + b\epsilon^2 + c\epsilon) \\
\Delta^2 f_i &= (6a\epsilon^2)t_i + (6a\epsilon^3 + 2b\epsilon^2) \\
\Delta^3 f &= 6a\epsilon^3
\end{aligned}$$

We moeten de startwaarden berekenen m.b.v. $t_0 = 0$:

$$\begin{aligned}
f_0 &= d \\
\Delta f_0 &= a\epsilon^3 + b\epsilon^2 + c\epsilon \\
\Delta^2 f_0 &= 6a\epsilon^3 + 2b\epsilon^2 \\
\Delta^3 f &= 6a\epsilon^3
\end{aligned}$$

Nu hebben we alles wat we nodig hebben, en kunnen we de interpolatiepunten zelf gaan berekenen als volgt:

$$\begin{aligned}
f_1 &= f_0 + \Delta f_0 \\
f_2 &= f_1 + \Delta f_1 = f_1 + \Delta f_0 + \Delta^2 f_0 \\
f_3 &= f_2 + \Delta f_2 = f_2 + \Delta f_1 + \Delta^2 f_1 = f_2 + \Delta f_1 + \Delta^2 f_0 + \Delta^3 f \\
f_4 &= f_3 + \Delta f_3 = f_3 + \Delta f_2 + \Delta^2 f_2 = f_3 + \Delta f_2 + \Delta^2 f_1 + \Delta^3 f \\
&\vdots \\
f_{i+1} &= f_i + \Delta f_{i-1} + \Delta^2 f_{i-2} + \Delta^3 f \quad i = 2, 3, \dots, m-1
\end{aligned}$$

2.2.3 C0-continuïteit

C0-continue curves zijn de standaard curves, vermits C0-continuïteit enkel de eis oplegt dat de curve ononderbroken moet zijn. Indien we als eerste controlepunt van een viertal het laatste controlepunt van het vorige viertal nemen, dan bekomen we automatisch C0-continuïteit. Deze eis vraagt dus geen veranderingen aan de controlepunten: het primitieve Bézier-algoritme voldoet (zie figuur 8).

2.2.4 G1-continuïteit

G1-continuïteit eist dat de curves naast ononderbroken ook overal vloeind ogen. Dit wordt dus bekomen door in elk controlepunt dat op de curve komt te liggen, de curve zó te berekenen dat de ingaande en de uitgaande raakvector in dat controlepunt op één lijn liggen (zie figuur 9). In Curve Editor is dit opgelost als volgt:

De ligging van het tweede en het derde controlepunt van een viertal Bézier-controlepunten wordt herberekend, zodanig dat de berekende curve vloeind aansluit op de berekende curve voor het vorige en het volgende viertal. Stel

het viertal v_1 en viertal v_2 opeenvolgende viertallen van controlepunten. Het laatste controlepunt van v_1 en het eerste controlepunt van v_2 zijn logischerwijs dezelfde: anders zouden we niet tot een aaneengesloten curve kunnen komen (zie C0-continuïteit). M.b.v. het voorlaatste controlepunt van v_1 , het gemeenschappelijke controlepunt en het tweede controlepunt van v_2 berekent Curve Editor nu een nieuw voorlaatste controlepunt voor v_1 en een nieuw tweede controlepunt voor v_2 (het gemeenschappelijke blijft ongewijzigd). Dit doet het door de vector tussen het voorlaatste controlepunt van v_1 en het tweede controlepunt van v_2 te beschouwen; die vector wordt dan zodanig verschoven dat het gemeenschappelijke controlepunt op die vector komt te liggen. De twee uiteinden van deze verschoven vector zijn dan het nieuwe voorlaatste controlepunt voor v_1 en het nieuwe tweede controlepunt voor v_2 . Indien we nu op die manier voor elk gemeenschappelijk controlepunt twee nieuwe controlepunten berekenen, dan bekomen we een mooi vloeiende curve.

2.2.5 C1-continuïteit

C1-continuïteit eist dat de curves naast ononderbroken zijn en overal vloeidend ogen, ook nog eens in elk controlepunt een gelijkmatig toe- en afnemende kromming hebben. Dit wordt dus bekomen door in elk controlepunt dat op de curve komt te liggen, de curve *zó* te berekenen dat de ingaande en de uitgaande raakvector in dat controlepunt op één lijn liggen én even groot zijn (zie figuur 10). In Curve Editor is dit opgelost als volgt:

De ligging van het tweede en het derde controlepunt van een viertal Bézier-controlepunten wordt herberekend, zodanig dat de berekende curve vloeiend aansluit op de berekende curve voor het vorige en het volgende viertal én zodanig dat de berekende curve een controlepunt even snel "binnenkomt" als "verlaat", met dezelfde versnelling als het ware. Stel het viertal v_1 en viertal v_2 opeenvolgende viertallen van controlepunten. Het laatste controlepunt van v_1 en het eerste controlepunt van v_2 zijn logischerwijs dezelfde: anders zouden we niet tot een aaneengesloten curve kunnen komen (zie C0-continuïteit). M.b.v. het voorlaatste controlepunt van v_1 , het gemeenschappelijke controlepunt en het tweede controlepunt van v_2 berekent Curve Editor nu een nieuw voorlaatste controlepunt voor v_1 en een nieuw tweede controlepunt voor v_2 (het gemeenschappelijke blijft ongewijzigd). Dit doet het door de vector tussen het voorlaatste controlepunt van v_1 en het tweede controlepunt van v_2 te beschouwen; die vector wordt dan zodanig verschoven dat het gemeenschappelijke controlepunt midden op die vector komt te liggen. De twee uiteinden van deze verschoven vector zijn dan het nieuwe voorlaatste controlepunt voor v_1 en het nieuwe tweede controlepunt voor v_2 . Indien we nu op die manier voor elk gemeenschappelijk controlepunt twee nieuwe controlepunten berekenen, dan bekomen we een mooi vloeiende en een in elk controlepunt even snel ingaand als uitgaand versnellende curve.

2.3 Hermite

3 Implementatie

3.1 Packages

3.1.1 Java packages

Een Java package is een mechanisme binnen Java om klassen te organiseren in namespaces. Java broncode die binnen eenzelfde categorie of functie vallen kunnen hierdoor gegroepeerd worden. Dit kan door middel van een package statement bovenaan het beginbestand om aan te geven waartoe ze behoren. Dit is omwille van twee redenen handig: de klassen zijn dan gegroepeerd in functionele categorieën, wat het geheel overzichtlijker maakt. En verder kunnen er nu twee verschillende klassen éénzelfde naam krijgen en toch uniek bepaald worden door er zijn package name voor te zetten. Dit is zeker handig als de programmeur een klasse dezelfde naam heeft gegeven als een klasse uit een library die hij wilt gaan gebruiken.

Voor Curve Editor zijn verschillende packages gemaakt, die zijn te zien op figuur 14. In wat volgt wordt een korte beschrijving gegeven van al deze packages, zonder ál teveel op de technische details in te gaan.

3.1.2 CurveEditor

Dit is wellicht de kleinste package van de reeks. Het bevat slechts één klasse, namelijk die klasse die de main-methode bevat. Deze klasse zal, zoals wellicht duidelijk is, als bootstrap dienen voor Curve Editor. Er wordt ook de mogelijkheid geboden om rechtstreeks vanuit de command line een bestand mee te geven. Dit is enkel ter volledigheid, vermits de gebruiker in de applicatie zelf zeer makkelijk bestanden kan inladen en opslaan.

3.1.3 Algorithms

Dit pakket voorziet de verschillende klassen die voor de interpolatiealgoritmen zullen zorgen. Elke klasse in dit pakket implementeert de Algorithm-interface. Dit is handig voor latere uitbreidingen en groepswerk: deze interface legt immers vast welke functies de programmeur zal moeten implementeren, die functies worden immers elders in de applicatie gebruikt.

De klassenamen zijn triviaal gekozen: “Lineair, Bezier, BezierC1, BezierG1, Hermite, HermiteCardinal, HermiteCatmullRom” (zie figuur 15). Zoals de namen al doen vermoeden, zullen deze klassen de verschillende interpolatiemethodes besproken in het deel ‘Wiskundige voorkennis’ (2) implementeren. Hierbij werd natuurlijk altijd geopteerd voor de meest optimale implementatie van degene die besproken werden.

3.1.4 Core

Dit pakket bevat enkele noodzakelijk klassen (zie figuur 16).

De klasse CurveContainer zal ervoor zorgen dat ingegeven punten kunnen opgeslagen worden, samen met hun door interpolatie berekende punten.

Een eerste implementatie gebruikte het subdivision principe. In de beginsituatie is het tekenveld dan één grote rechthoek. Van zodra de gebruiker een curve begint te tekenen worden de secties, waar nieuwe punten geplaatst zijn, onderverdeeld in steeds kleiner wordende rechthoekjes. In elk zo'n rechthoekje zat dan juist één punt van een curve. Nagaan welke curve op een bepaalde pixel lag, was op deze manier makkelijk te achterhalen m.b.v. een quadtree.

Deze implementatie bleek echter niet zo efficiënt te zijn wanneer we te maken hadden met een groot aantal controlepunten. Dit kwam voornamelijk doordat er telkens opnieuw kleinere rechthoekjes moesten aangemaakt worden bij het ingeven van een nieuw punt, en een grotere wanneer er punten verwijderd werden. Uiteindelijk was de applicatie meer bezig met het bepalen van rechthoekjes dan met zijn doel: bijnouden en picken van curves. De gebruikte datastructuur gaf wel mooie resultaten voor het zoeken van curves a.h.v. een geklikt of gehooverd punt, maar wanneer de gebruiker de curves ging aanpassen (nieuwe punten, verplaatsen, connecteren, ...), dan ging de herberekening en aanpassing van de datastructuur echter tergend langzaam. Wij achdden het dan ook noodzakelijk dat we een andere manier zouden vinden, die een betere balans tussen aanpassen en informatie halen uit”.

De tweede door ons geïmplementeerde datastructuur leek ons direct bruikbaarder. Het aanpassen van de datastructuur ging immers direct een pak sneller in vgl. met de vorige implementatie, de gebruiker kan nu bvb. redelijk vlug nieuwe controlepunten vlak achter elkaar ingeven. Ook het halen van informatie uit de datastructuur heeft een aanvaardbare snelheid. Wat stelt die datastructuur nu precies voor ? We stellen een tweedimensionale matrix op waarvan elk element een referentie kan bevatten naar een curve-instantie. Het toevoegen van curves is op deze manier zeer simpel te doen: voor elk punt van de curve plaatsen we in de overeenkomstige cel van de matrix een referentie naar de curve. Curves verwijderen is dan gewoon het tegenovergestelde: al die cellen weer naar null laten wijzen. Het zoeken naar een curve a.h.v. een punt (a.h.v. coördinaten van een muisevent, bijvoorbeeld) gaat nog vrij snel omdat we enkel maar in een bepaalde range in de matrix de inhoud van de cellen moeten bekijken. Indien een niet-null referentie in die range aanwezig is, dan weten we welke curve daar ligt.

De klasse Editor is het hart van Curve Editor. Deze zal zorgen dat ge-

gevens tussen de verschillende datastructuren (Point, Curve, Algorithm) kan doorgegeven worden. De uitwisseling van data zal voornamelijk bestaan uit het zoeken/verwijderen van punten uit de CurveContainer-klasse en het veranderen van Curves (verplaatsen, punten toevoegen/verwijderen, type veranderen, ...). Maar ook het opvangen en afhandelen van exceptions gebeurt grotendeels in Editor. Deze klasse zorgt er m.a.w. voor dat de verschillende andere klassen zo autonoom mogelijk kunnen werken, Editor is dan de klasse die alles samenbrengt. Dit verhoogt natuurlijk ook de leesbaarheid en onderhoudbaarheid van de code.

Een laatste klasse van dit pakket is de FileIO-klasse, deze zal niet alleen bestanden opslaan en inladen, maar ook zal hij de undo/redo-functionaliteit, vermits deze van dezelfde functies gebruik maakt. Er wordt dan gewoon een stack van bestanden in het geheugen bijgehouden.

3.1.5 Curves

Dit pakket bevat de twee datatypes die doorheen het programma gebruikt worden (zie figuur 17). Point verzorgt, zoals de naam doet vermoeden, de datastructuur die punten voorstelt. Curve geeft dan weer de mogelijkheid om een verzameling van punten (lees: een kromme of curve) op te slaan. De technische details van deze laatste klasse worden beter uitgelegd verder in de tekst (zie). Voorlopig is het voldoende om te weten dat Curve een vector van Point-instanties bijhoudt en enkele basisvoorzieningen voorziet (punten opvragen, toevoegen, translateren, ...).

3.1.6 Exceptions

Een foutloos programma schrijven is op zich al een opgave, vermits er altijd wel kleine bugs kunnen opduiken na langdurig en gevareerd gebruik. Foolproof code schrijven daarentegen is een onmogelijke opgave. Daarom hebben we gebruik gemaakt van exceptions om “verkeerd” gebruik van CurveEditor-functies op te vangen (zie figuur 18). Onder verkeerd gebruik valt bijvoorbeeld het inladen van een onbestaande file of een verkeerd bestandsformaat, het toevoegen van een punt zonder er de coördinaten van op te geven, Maar ook met toekomstige uitbreidingen in gedachte zijn exceptions een handig middel, niet elke programmeur hanteert immers dezelfde logica. Vindt de één dat aan een methode bijvoorbeeld geen null-waarden kunnen worden meegegeven omdat ’t nonsens lijkt, dan kan een ander vinden dat het wel mogelijk moet zijn om dat te doen. Exceptions laten die andere dan weten dat zo’n parameters nooit mogen meegegeven worden wil hij een resultaat verkrijgen, zonder dat hij naar de broncode van die ene methode hoeft te kijken.

Er zijn ook twee HandleException-klassen voorzien (zie figuur 19): eentje in dit pakket, deze zal gewoon het exceptie bericht in de console uitprinten, en eentje in het GUI-pakket, die in een dialoogvenstertje de exceptie zal uitprinten.

3.1.7 GUI

De GUI is vrij grondig opgesplitst (zie figuur 20). Elk groot deel heeft zo zijn eigen klasse; zo heb je een menubalk-klasse, een snelknopbalk-klasse, een tekengebied-klasse en een keuzegebied-klasse.

De GUI-klasse zal op zijn beurt het centrale orgaan spelen die al deze klassen met elkaar verbindt. Het is daarom ook logisch dat deze is afgeleid van de klasse Editor, de centrale klasse van de Core-package. Zo kunnen immers Core-functionaliteiten aan GUI-events e.d. gelinkt worden.

Het opdelen van de GUI in verschillende klassen heeft enorme voordelen als we spreken over leesbaarheid en onderhoudbaarheid van code. Het EventHandling-systeem van Java zelf is overigens optimaal voor dit soort afscheiding. In Java worden de events immers opgevangen in een klasse, die afgeleid wordt van een van de Listener-interfaces (ActionListener, ItemListener, ...). Deze klasse zal dan de door de interface opgelegde functies implementeren om also de juiste evenafhandeling te kunnen doen. Het is dus voldoende om deze EventHandling-klassen in GUI aan te maken en door te geven aan de juiste componenten (menubalk, tekengebied, ...). Meer uitleg hierover vind je verder in de tekst.

3.1.8 Tools

Dit bevat een curvesimulator (zie figuur 21). In principe is het een “chique” naam voor een bolletje dat over het pad dat de curve voorstelt loopt. Dit kan bijvoorbeeld door de gebruiker gebruikt worden om de stijging en daling van de curve te bestuderen. Zodoende kan hij de getekende curve visueel evalueren.

3.2 Uitwerking van de GUI

3.2.1 Overzicht

Zoals in figuur 3 te zien is kan de vormgeving van de GUI in 4 grote stukken worden opgedeeld:

1. Menubalk

Hierin kan de gebruiker elke actie terugvinden die door Curve Editor kan gedaan worden. Bijna elke actie is ook aan te roepen met behulp van sneltoetscombinaties (Alt-F, Ctrl-O, ...), zodat elke gebruiker op zijn favoriete manier kan navigeren doorheen de verschillende uitvoerbare taken. Er is overigens aandacht geschenken aan de nesting van de

menu-items: die mogen niet te diep noch te breed zijn. Verder is elke naam zo triviaal mogelijk gekozen en meestal wordt die ook voorzien van een icoontje (zie figuur ??).

2. Snelknopbalk Hierin vind je enkele veelgebruikte taken terug (nieuw bestand beginnen, bestand openen, nieuwe curve starten, ...). Dit is voorzien opdat de gebruiker deze taken snel en dus efficiënter kan afhandelen dan wanneer hij dat telkens vanuit de menubalk zou moeten doen.
3. Keuzegebied

In dit gebied zal de gebruiker snel kunnen kiezen welk type curve hij wilt tekenen (Bézier, Hermite, ...) en kan hij ten alle tijden een punt aan de op dat moment geselecteerde curves toevoegen door de coördinaten in te geven. Verder kan de gebruiker hier ook kiezen of hij de coördinaten, raakvectoren en puntnummers van de geselecteerde curves wilt laten uittekenen (zie figuur 4).

Het Edit-veld van dit gebied hangt af van de mode waarin je bezig bent. Er zijn voor de gebruiker twee grote modi "voelbaar". De eerste is curve-modus: hier bewerk je de curves op zich (selecteren, deselecteren, nieuwe curve maken, ...); de tweede is controlepunt-modus (punt verslepen, selecteren, toevoegen). Naargelang de modus waarin je zit zullen andere mogelijkheden zichtbaar worden in het Edit-veld. Dit leidt tot een groter gebruiksgemak vermits de taken die verschijnen meestal ook gebruikt worden wanneer je in die modus werkt.

4. Tekengebied

Dit gebied is waarschijnlijk wel het interessantste voor de gebruiker. Met behulp van muisinteractie kan de gebruiker controlepunten aan dit veld toevoegen en deze punten laten connecteren door een interpolatiealgoritme. De gebruiker kan tevens in dit veld punten en curves selecteren, verwijderen en zelfs verslepen. Het tekengebied maakt gebruik van double buffering om beeldflikkering te vermijden en ook van clipping zodat alleen getekend wordt wat ook echt op het scherm kan verschijnen.

3.3 Listeners in Java

(zie figuur 22) Java heeft een aparte stijl om events op te vangen (signalen gestuurd door buttons, menu-items, ...). De taal bezit hiervoor zogenaamde EventListener-interfaces. De programmeur moet een klasse aanmaken die een dergelijke interface zal implementeren. Deze klasse wordt dan m.b.v. een simpel commando geconnecteerd met een bepaalde component (bijvoorbeeld een button). In de interface is er een EventHandle-functie gedefinieerd die door de programmeur zal geïmplementeerd worden, zodat, wanneer de

component een event genereert, die opgevangen en afgehandeld kan worden.

Dit systeem is pas echt voordelig als je een event moet afhandelen in een geneste klassestructuur. Stel als voorbeeld: je hebt in een klasse A een instantie van een klasse B zitten die op zijn beurt dan weer een instantie heeft van klasse C, en je wilt dat klasse A een handeling doet als klasse C een event uitstuurt. Met callbacks is de properste manier (om klasse-onafhankelijkheid te bewaren) een callback te connecteren van klasse A naar klasse B en daarna een callback van klasse B naar klasse C. In Java kan je gewoon een EventHandle-klasse vanuit klasse A meegeven aan B die het op zijn beurt meegeeft aan klasse C. Dit maakt de structuur natuurlijk wat eenvoudiger, overzichtelijker, en makkelijker te begrijpen.

In deze implementatie is ervoor gekozen om deze EventHandle-klassen in de klasse GUI te definiëren. Het voordeel hiervan is dat enkel de klasse GUI de EventHandle-klassen kan instantiëren en deze instanties zullen alle membervariabelen en -functies van GUI zelf kunnen gebruiken. Dit zorgt ervoor dat er geen extra connectie moet gelegd worden tussen de EventHandle-klassen en GUI. Je kan dit dus bekijken als een soort van friendly class die niet geïnstantieerd kan worden.

3.4 Datastructuren

(zie figuur 17)

3.4.1 Point

Deze datastructuur heeft niet zoveel uitleg nodig, een Point bestaat namelijk uit zijn x- en y-waarde en is daarmee volledig gedefinieerd. De nodige get()- en set()-functies zijn ook voorzien, uiteraard.

3.4.2 Curve

Deze structuur heeft al iets meer uitleg nodig. Een curve is, vanuit het laagste niveau bekeken, niets anders dan een container van zijn punten. Elke Curve houdt drie essentiële dingen bij

1. zijn type (Bézier, Hermite, ...)
2. een Vector van controlepunten: deze wordt dus gevuld wanneer de gebruiker een nieuw controlepunt aan de curve toevoegt
3. een Vector van outputpunten: deze wordt a.h.v. de Vector van controlepunten en m.b.v. een Algorithm-instantie gevuld

Hierbij worden het type en de controlepunten dus door de gebruiker ingegeven/veranderd. De vector van outputpunten wordt intern berekend door de verschillende interpolatiealgoritmen. Het aanroepen van deze Algorithmfuncties gebeurt niet in Curve, maar in Editor, de klasse die alle datastructuren samenbrengt. De vectoren worden ook door de GUI gebruikt om de curve op het tekengebied te kunnen uittekenen.

Om de punten op te slaan hebben we voor een Vector gekozen, omdat de verschillende interpolatietechnieken gemakkelijk moeten kunnen "springen" tussen de gegeven controlepunten om hun algoritme te kunnen uitvoeren.

Deze klasse bevat ook nog functies om punten te verwijderen, toe te voegen, om na te gaan of een gegeven punt al dan niet tot de curve behoort,

3.5 Extra's

Echte extra's hebben we niet geïmplementeerd, onderstaande uitgezonderd. Bestanden opslaan naar meerdere formaten (*.xml en *.gif/*.png), bestanden inladen, oneindige undo/redo-stack, meer performante code verkiezen, selectietooltje m.b.v. dragging (zie figuur 5), ... beschouwen we gewoon als "nodig" voor Curve Editor.

3.5.1 Path Simulation Tool

Dit tooltje loopt de geselecteerde curves één voor één af, en laat een bolletje verschijnen dat als het ware "loopt" van het begin naar het einde van de curve. Het starten van deze tool, laat een nieuwe thread starten, zodat men één de tool kan laten runnen én men terwijl de geselecteerde curves zelf nog kan aanpassen (verslepen e.d.).

3.6 Interessante problemen

3.6.1 Vector vs. LinkedList

Hoewel de collecties van outputpunten van curves enkel maar sequentiel worden doorlopen (toevoegen achteraan, hele inhoud verwijderen, van voor naar achter uittekenen, ...), bleek Vector toch de betere keuze te zijn. Hoewel de structuur van LinkedList ook zou moeten voldoen, maar met een kleiner geheugenverbruik dan Vector, bleek deze datastructuur tergend langzaam te werken. Een reden hiervoor hebben we niet gevonden, maar door gewoon terug gebruik te maken van Vector was het probleem ook opgelost.

3.6.2 HashMap vs.2D Matrix

Voor de CurveContainer, die de picking moet afhandelen, hadden wij aanvankelijk een HashMap gebruikt. De keys van de HashMap in de bovenste

laag waren de pixels/punten van het scherm, en de values waren referenties naar rechthoekjes. Bij initialisatie van de CurveContainer wijzen alle values naar eenzelfde rechthoek. Elk van die rechthoekjes bevat één referentie naar een Curve-instantie en een Point-instantie, of twee null-referenties. Zo wordt elk rechthoekje gemapt op een Curve.

Wordt er een extra punt aan de container toegevoegd, dan wordt de rechthoek naarwaar de pixel momenteel verwijst opgesplitst in vier deelrechthoekjes, indien die niet naar null verwijst en indien de resulterende rechthoekjes niet té klein zijn. Indien die wel naar null refereert, dan wordt de hele rechthoek gemapt op die Curve zonder de rechthoek daarvoor op te splitsen, en wordt de Point-referentie gelijkgesteld aan de referentie naar het nieuwe controlepunt. Als er vier deelrechthoekjes aangemaakt dienen te worden, dan wordt voor elk van die rechthoekjes bepaald naar welke Curve zij moeten verwijzen, die van het nieuwe controlepunt, of de Curve waarnaar de grote rechthoek refereerde. Het rechthoekje waarin het nieuwe controlepunt ligt, wordt automatisch aangepast zodat het naar de andere Curve verwijst. Voor de 3 andere wordt de afstand berekend van het middelpunt van het rechthoekje tot het punt naarwaar de Point-instantie van de omkoepelende rechthoek wijst. Is die afstand groter dan de afstand van het middelpunt tot het nieuwe controlepunt, dan zal dat rechthoekje niet de referenties van de omkoepelende rechthoek overnemen, maar naar de nieuwe Curve/Point refereren. Alle punten in dat rechthoekje wijzen in de HashMap naar dat rechthoekje, uiteraard.

Een curve verwijderen gaat als volgt: men gaat voor alle punten de hashmap af, indien het gevonden rechthoekje naar de curve wijst, dan dient dat rechthoekje aangepast te worden. Als de omliggende rechthoekjes naar eenzelfde curve verwijzen, dan wordt van die vier rechthoekjes weer n rechthoek gemaakt. Indien de omliggende rechthoekjes niet naar eenzelfde curve verwijzen, dan wordt weer de afstand berekend van het middelpunt van het te veranderen rechthoekje tot de punten naarwaar de andere rechthoekjes refereren. Hetgeen de kleinste afstand oplevert, heeft de referenties die men zoekt.

Een Curve opzoeken gaat razendsnel doordat gewoon in de bovenste HashMap de value voor de key/punt gehaald moet worden. A.h.v. het bekomen rechthoekje kan men weten naar welke Curve het punt wijst.

Jammer genoeg ging het telkens aanpassen van de hele rechthoekstructuur zeer langzaam wat tot ergernissen bij de eindgebruiker zou leiden. Het picking ging echter wel héél vlug, maar we verkiezen toch een datastructuur die misschien net iets minder vlug is voor picking, maar voor aanpassingen aanvaardbare snelheden geeft. Een eerste datastructuur die we implementeerden was een simpele Matrix zoals die uitgelegd werd in sectie 3.1.4. Die bleek vlot genoeg te werken, dus uiteindelijk hebben we deze datastructuur maar behouden.

3.6.3 Systeemafhankelijke problemen

Op de meeste PC's werkt Curve Editor zoals het hoort, maar op sommige systemen blijkt het toch minder performant te werken.

Zo werkt het pixel-tekenen van Curve-Editor op UHasselt-laptops zéér traag. Dit zal waarschijnlijk te wijten zijn aan slechte Nvidia-drivers, want op diezelfde laptops werkt het in Linux wel vlotjes. Op andere geteste Windows-systemen doken er geen problemen op.

Een foute configuratie van de Xorg-server in Linux zorgt ook voor problemen. Zo is het op één getest systeem onmogelijk Curve Editor te resizen of te verplaatsen. Dat laatste lukt wel, maar het zorgt ervoor dat wanneer je dan op menu-items klikt, ze direct terug "wegen".

4 Planning

12/02/08	Maken en afgeven van: beginverslag en planning van de komende dagen	Maken en afgeven van: beginverslag en planning van de komende dagen	13/02/08
14/02/08	Opzoekwerk java mogelijkheden (java swing, eventlisteners, java en 2d tekenen, ...)	Opzoekwerk java mogelijkheden (java swing, eventlisteners, java en 2d tekenen, ...)	15/02/08 Opzoekwerk java mogelijkheden (java swing, eventlisteners, java en 2d tekenen, ...)
16/02/08		Opzoekwerk java mogelijkheden (java swing, eventlisteners, java en 2d tekenen, ...)	17/02/08
18/02/08			19/02/08 Feedback afwachten
20/02/08	Feedback afwachten	Feedback afwachten	21/02/08 Samenbrengen van opzoekresultaten
22/02/08	23/02/08		

Samenbrengen van opzoekresultaten	Samenbrengen van opzoekresultaten Een dag bleek hier voor genoeg te zijn.	Feedback bespreken	Feedback bespreken		
24/02/08	25/02/08				
Aanpassingen maken in de structuur van het programma a.d.v. de feedback.	Aanpassingen maken in de structuur van het programma a.d.v. de feedback. Functionaliteit van het programma herzien.	Aanpassingen maken in de structuur van het programma a.d.v. de feedback.	Aanpassingen maken in de structuur van het programma a.d.v. de feedback.		
26/02/08		27/02/08 - 29/02/08			
Bespreken van de exacte taakverdeling. Bespreken van verdere taakplanning. Implementatie van het algorithme interface.	Bespreken van de exacte taakverdeling. Bespreken van verdere taakplanning. Implementatie van het algorithme interface.				
01/03/08 - 07/03/08					
voorbereiden examens					
08/03/08 - 21/03/08					
blok					
22/03/08 - 25/03/08					

uitrusten van de examens		
26/03/08	27/03/08	
Eerste bespreking van GUI	Eerste bespreking van GUI nieuwe ideeën voorleggen (path simulation, bestanden opslaan als xml file, mogelijkheid tot screenshot, ...)	
28/03/08	Tweede bespreking van de GUI en de nodige functionaliteit van het programma	Tweede bespreking van de GUI en de nodige functionaliteit van het programma uitgesteld tot 30/03/08
30/03/08	Tweede bespreking van de GUI en de nodige functionaliteit van het programma.	<i>Sibrand:</i> Begin implementatie van tekengebied, verder opzoekwerk naar specifieke methode, klassen en tekenmogelijkheden. <i>Sibren:</i> Begin implementatie van menu en keuzegebied. poging tot eventlisteners
01/04/08		<i>Sibrand:</i> Begin implementatie van tekengebied, verder opzoekwerk naar specifieke methode, klassen en tekenmogelijkheden. <i>Sibren:</i> Begin implementatie van menu en keuzegebied. <i>Sibren:</i> Niet toegekomen aan het implementeren van het keuzegebied.
		02/04/08

	<p>Herziening van de mockup aan de hand van enkele implementatie details.</p>	
03/04/08	04/04/08	
	<p><i>Sibrand:</i> verdere implementatie van het tekengebied (minstens rechte lijnen kunnen teken). <i>Sibren:</i> Menu en keuzegebied, gebruiks en klikbaar maken</p>	<p><i>Sibrand:</i> verdere implementatie van het tekengebied (minstens rechte lijnen kunnen teken). <i>Sibren:</i> Menu en keuzegebied, gebruiks en klikbaar maken <i>Sibren:: Menu 80% klaar en keuzegebied 40% klaar.</i></p>
05/04/08 - 14/05/08	 	
15/04/08	<p><i>Sibrand:</i> Implementatie van verschillende Bezier algorithmes. <i>Sibren:</i> Implementatie van verschillende Hermite algorithmes.</p>	16/04/08
	<p><i>Sibrand:</i> Implementatie van verschillende Bezier algorithmes. <i>Sibren:</i> Implementatie van verschillende Hermite algorithmes. <i>Sibrand:</i> Bezier en Bezier C0 volledig afgewerkt en getest. <i>Sibren:</i> Hermite en Hermite Cardinal afgewerkt en getest.</p>	
17/04/08		18/04/08

<i>Sibrand</i> : Implementatie van verschillende Bezier algorithmes. <i>Sibren</i> : Implementatie van verschillende Hermite algorithmes.	<i>Sibrand</i> : Implementatie van verschillende Bezier algorithmes. <i>Sibren</i> : Implementatie van verschillende Hermite algorithmes. <i>Sibrand</i> : Bezier C1 en Bezier G1 volledig afgewerkt en getest. <i>Sibren</i> : Hermite Catmull-Rom afgewerkt en getest, keuzegebied voor 80% afgewerkt.	
19/04/08	20/04/08	
	<i>Sibrand</i> : Tekengebied uitbreiden zodat bezier en hermite tekenen mogelijk zijn. <i>Sibren</i> : Implementatie van een toolbar	<i>Sibrand</i> : Tekengebied uitbreiden zodat bezier en hermite tekenen mogelijk zijn. <i>Sibren</i> : Implementatie van een toolbar
21/04/08	22/04/08	
 		
23/04/08	24/04/08	
	<i>Sibrand</i> : datastructuur voor het opslaan van curves uitwerken (deel 1)	<i>Sibrand</i> : datastructuur voor het opslaan van curves uitwerken (deel 1)
25/04/08	26/04/08	
 	<i>Sibren</i> : FileIO implementeren (gebruik makende van xml)	<i>Sibren</i> : FileIO implementeren (gebruik makende van xml)

27/04/08	28/04/08		
		Herziening van opslag datastructuren om performantie redenen	
29/04/08	30/04/08		
 			
01/05/08	02/05/08		
<i>Sibrand en Sibren:</i> gevonden "bugs" uit de code halen.	<i>Sibrand en Sibren:</i> gevonden "bugs" uit de code halen. Bepaalde GUI knoppen hernoemen, functionaliteit aan de gui toevoegen, Nog een laatste bespreking van de datastructuur om curves op te slaan.	<i>Sibrand:</i> Afwerken van datastructuur voor het opslaan van curves, begin implementeren algoritme voor curve/punt zoeken <i>Sibren:</i> Afwerken van FileIO, volledige afwerken implementatie van menu en toolbar.	<i>Sibrand:</i> Afwerken van datastructuur voor het opslaan van curves, begin implementeren algoritme voor curve/punt zoeken <i>Sibren:</i> Afwerken van FileIO, volledige afwerken implementatie van menu en toolbar. Iconen in menu en toolbar zijn nog niet aanwezig.
03/05/08 - 06/05/08			
 			
07/05/08	08/05/08		
	<i>Sibrand:</i> Volledige afwerking van van algoritmes i.v.m. punt - en curvebewerkingen <i>Sibren:</i> Full screen uitbreiding, herziening van Hermite algoritme. Verschillende code gedeelte juist connecteren.	<i>Sibrand:</i> Volledige afwerking van van algoritmes i.v.m. punt - en curvebewerkingen <i>Sibren:</i> Full screen uitbreiding, herziening van Hermite algoritme. Verschillende code gedeelte juist connecteren.	

		<i>Sibrand</i> : De performantie van de verschillende algoritmes is nog niet optimaal <i>Sibren</i> : Full screen werkt niet onder eender welke configuratie
09/05/08	10/05/08	
<i>Sibren</i> : Prototype verslag maken.	<i>Sibren</i> : Prototype verslag maken.	
11/05/08	12/05/08	
 	Debuggen en testen	Enkele bugs verwijderen, curve opslaan herzien om performantie redenen.
13/05/08	14/05/08	
 		
15/05/08	16/05/08	
Beëindigen van de het implementatie gedeelte van het project.	<i>Beëindigen van de het implementatie gedeelte van het project. Project is voor 90% af, nog een paar onigheden implementeren over de implementatie en GUI functionaliteit uitklaren. uitgesteld tot 18/05/08</i>	
17/05/08	18/05/08	

	<i>Sibrand en Sibren:</i> implementatie deel is afgerond.		deadline gehaald.
19/05/08		20/05/08	
21/05/08		22/05/08	
		Verslag, testen van programma en eventuele bugs verwijderen.	Verslag, testen van programma en eventuele bugs verwijderen.
23/05/08 - 28/05/08			
 			
29/05/08		30/05/08	
verslag en powerpoint presentatie.	<i>verslag.</i> <i>Powerpoint presentatie. (nog niet aan toegekomen).</i>		
31/05/08 - 03/05/08			

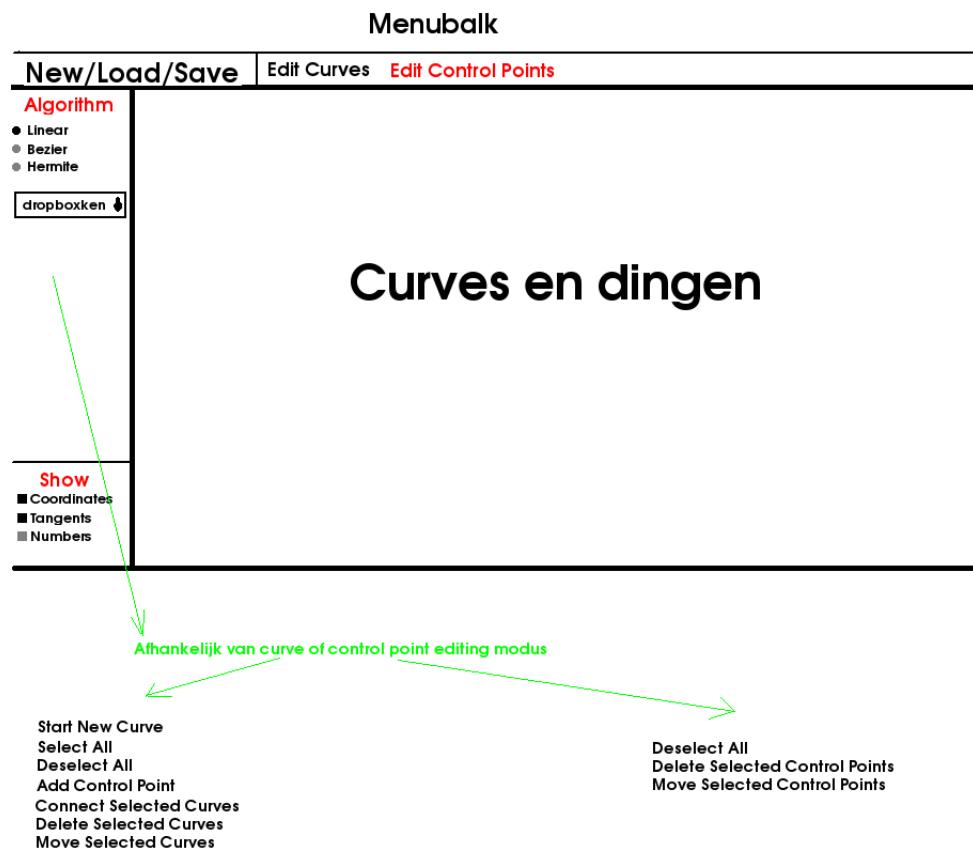
04/06/08	05/06/08	
	<i>Sibren</i> : Laatste blik op het verslag, code, werking van het programma. <i>Sibrand en Sibren</i> : Voorbereiding van de presentatie	
06/06/08 - 11/06/08		
12/06/08	13/06/08	
Presentatie		

5 Taakverdeling

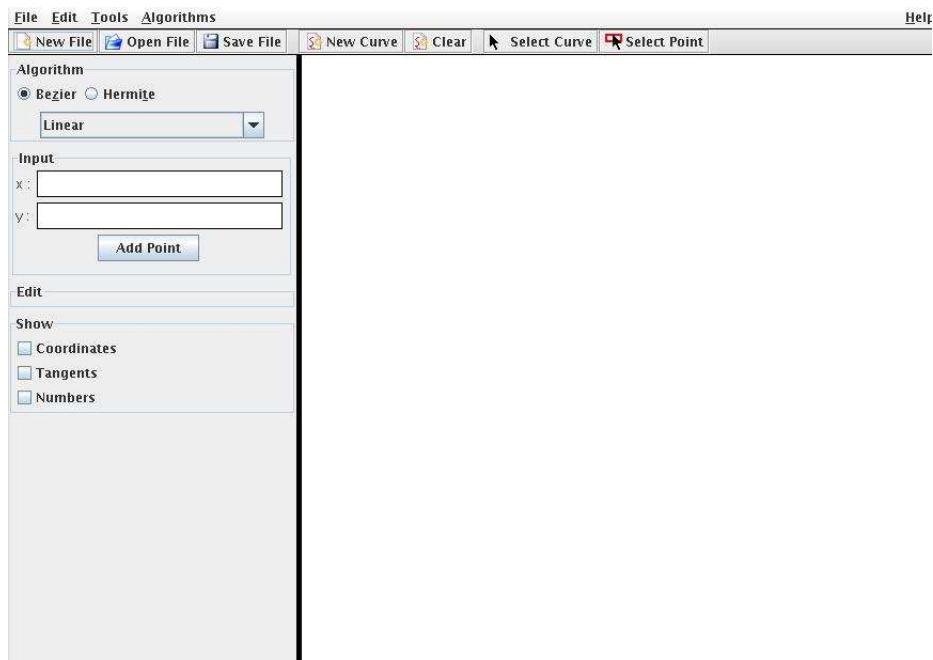
1. Sibrand Staessens: Bezier algorithmes, CurveContainer, Editor, DrawArea, GUI
2. Sibren Polders Hermite algorithmes, FileIO, Menu, ToolBar, ChoiceArea, GUI, PathSimulation

6 Appendix

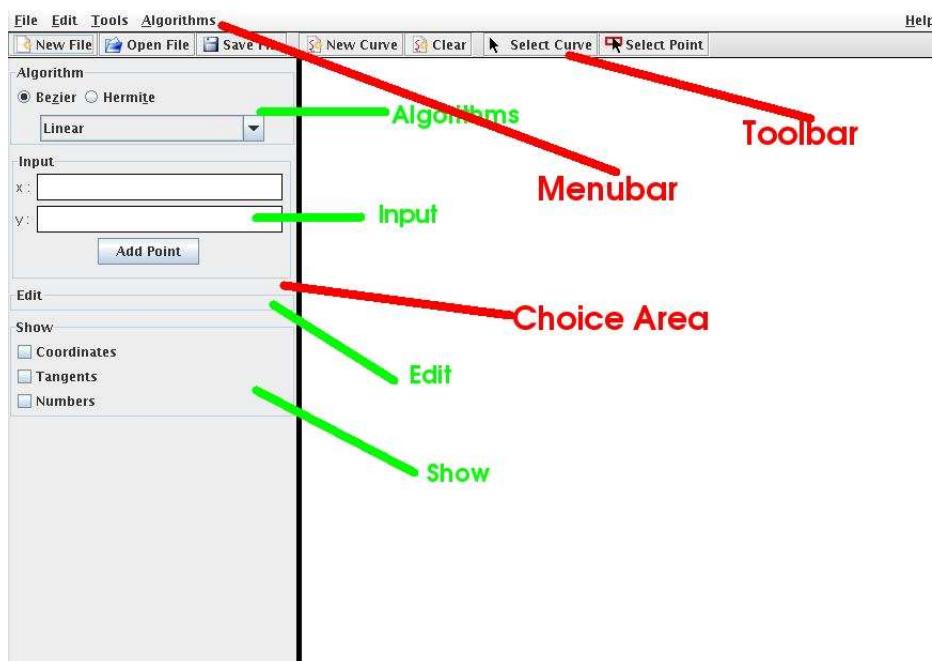
A Screenshots



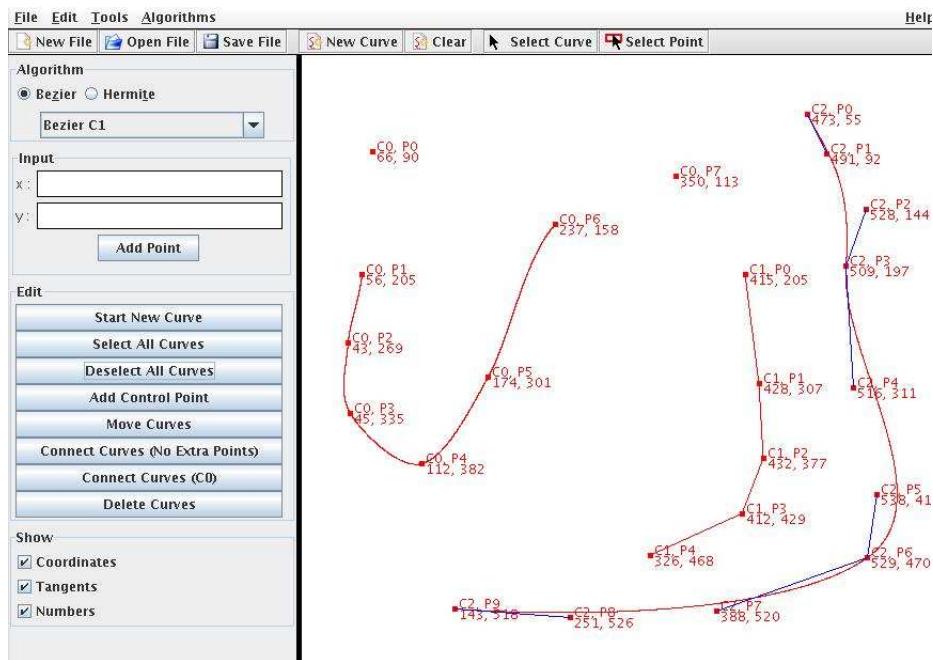
Figuur 1: Mock-up



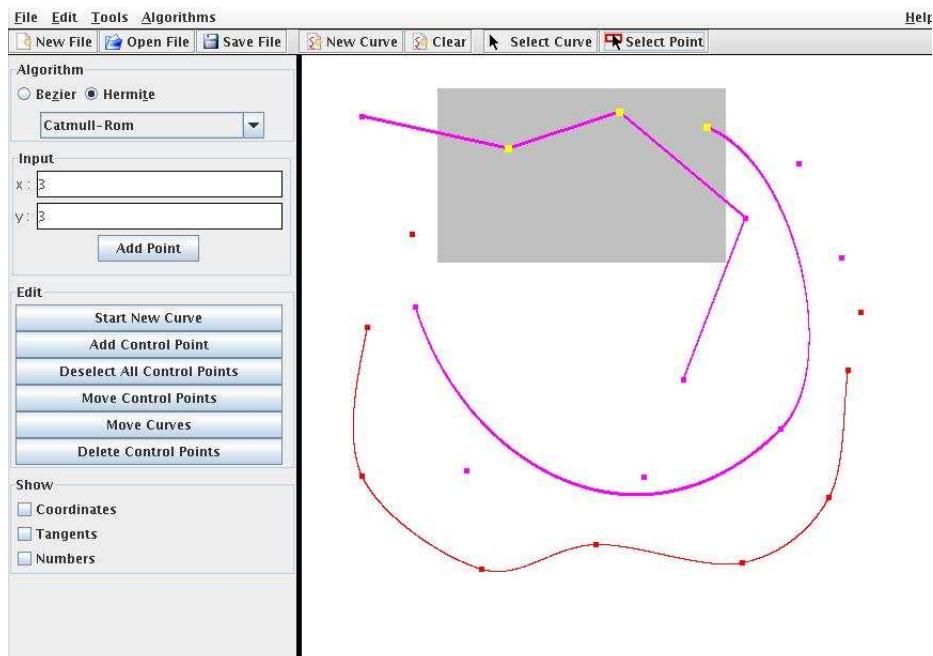
Figuur 2: Curve Editor-omgeving



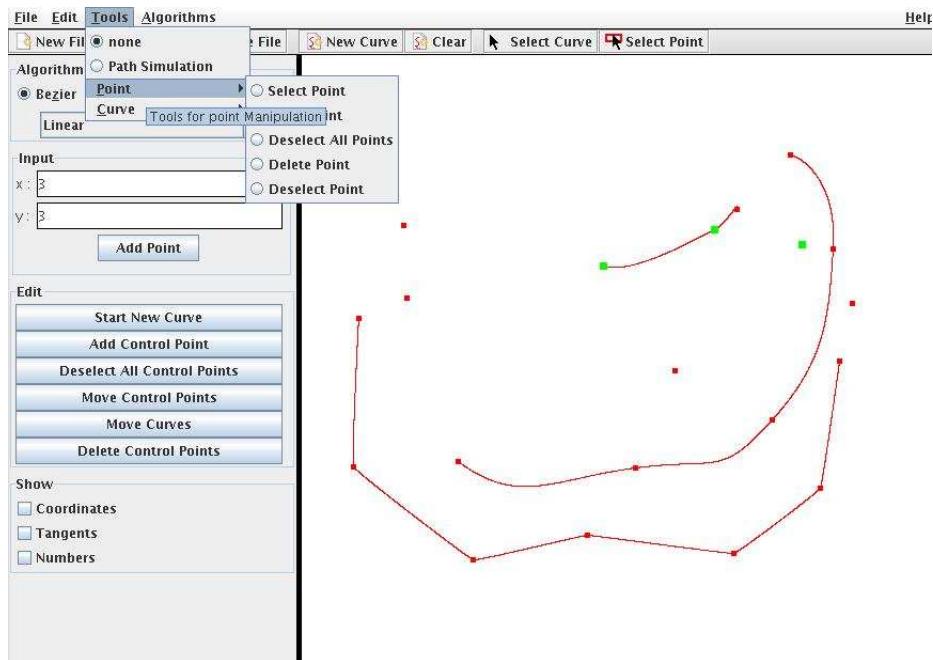
Figuur 3: Curve Editor-omgeving, aangeduid



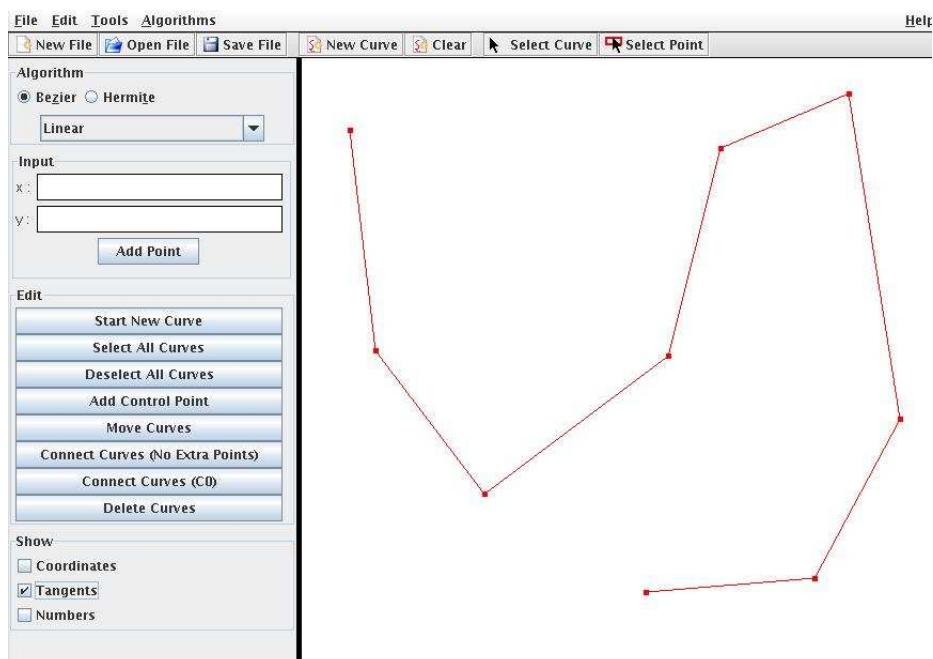
Figuur 4: Curve Editor-omgeving, alle weergaveopties getoggled



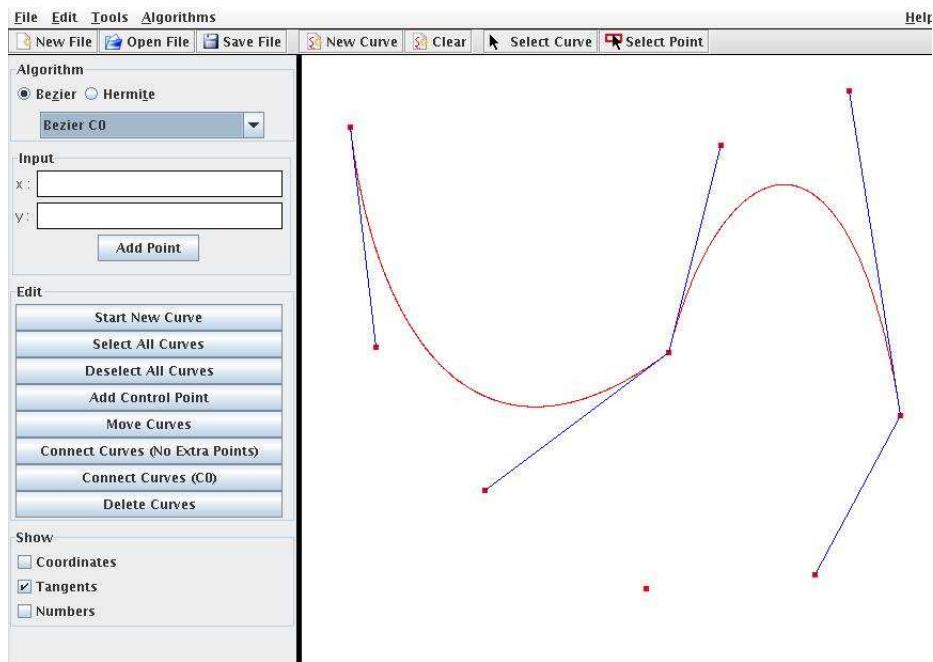
Figuur 5: Curve Editor-omgeving, dragging selectietooltje



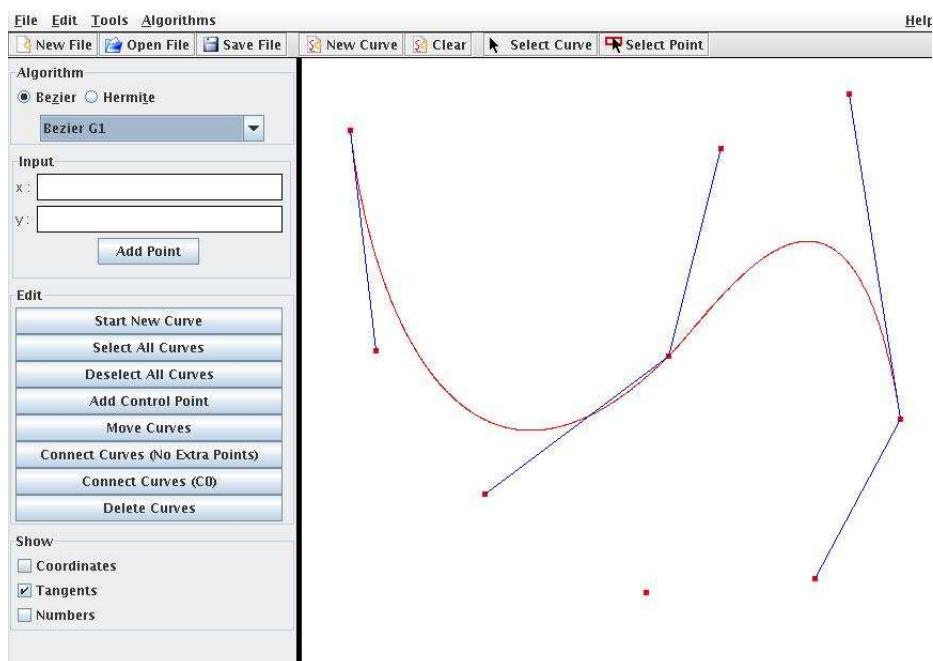
Figuur 6: Curve Editor-omgeving, Menu - Tools



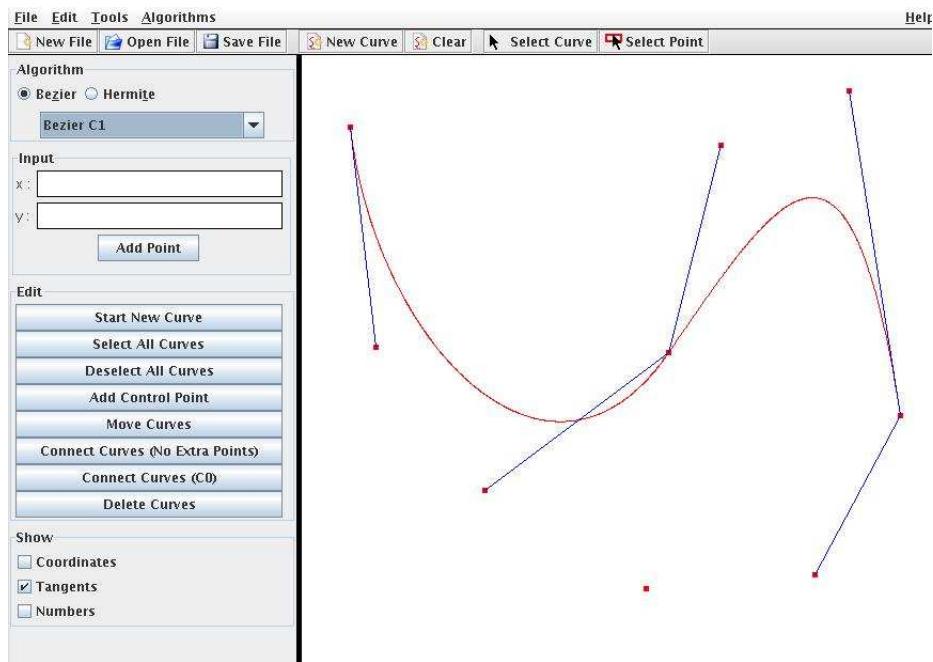
Figuur 7: Lineair



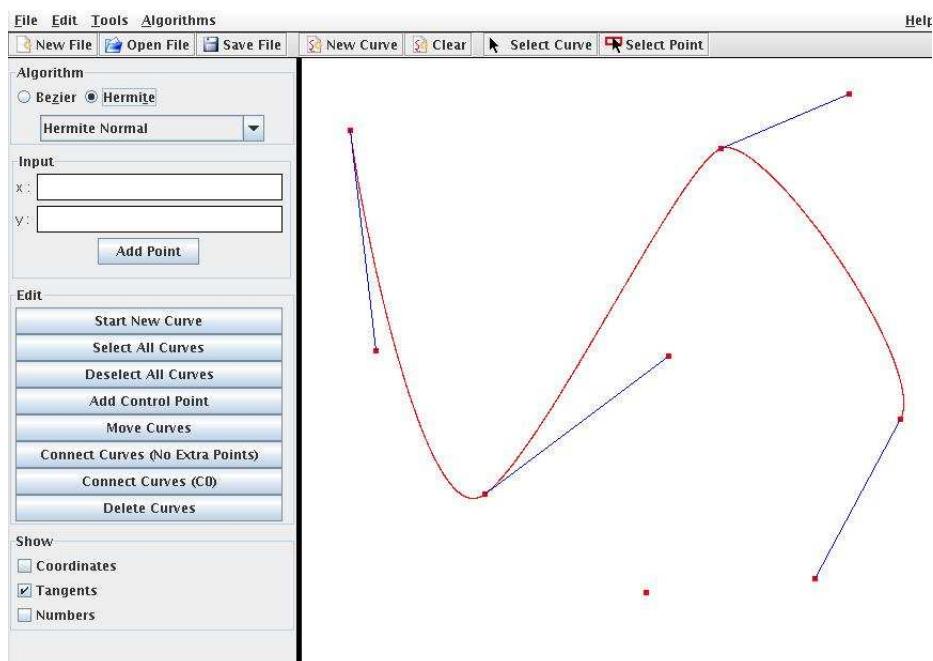
Figuur 8: Bézier, C0-continu



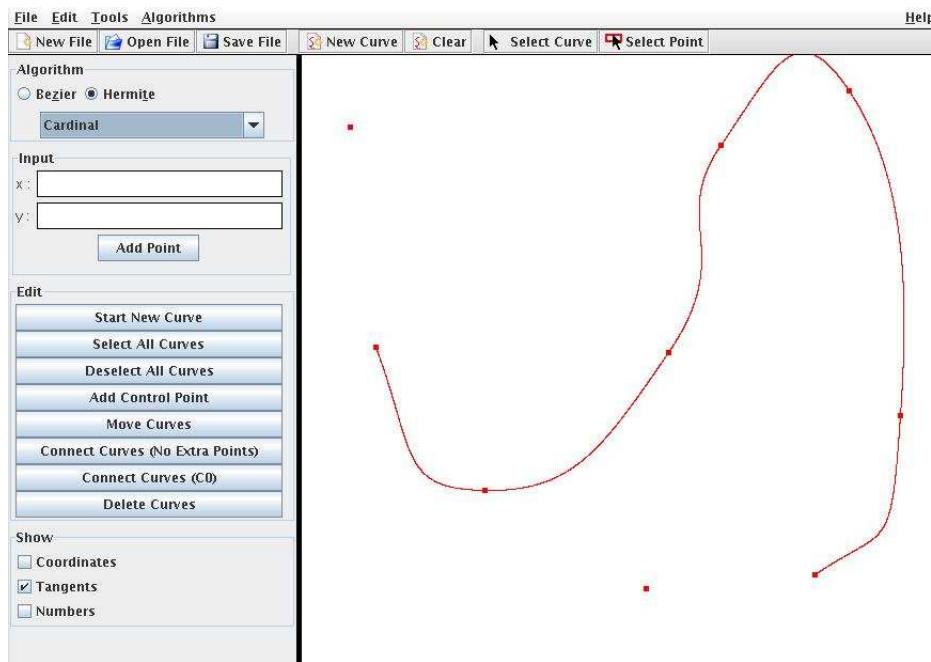
Figuur 9: Bézier, G1-continu



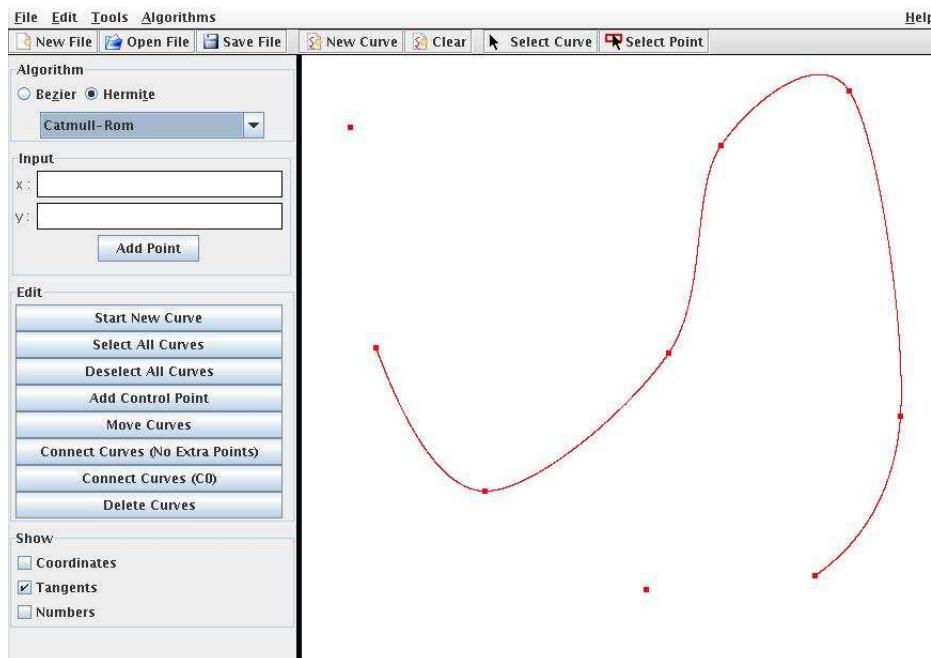
Figuur 10: Bézier, C₁-continu



Figuur 11: Hermite

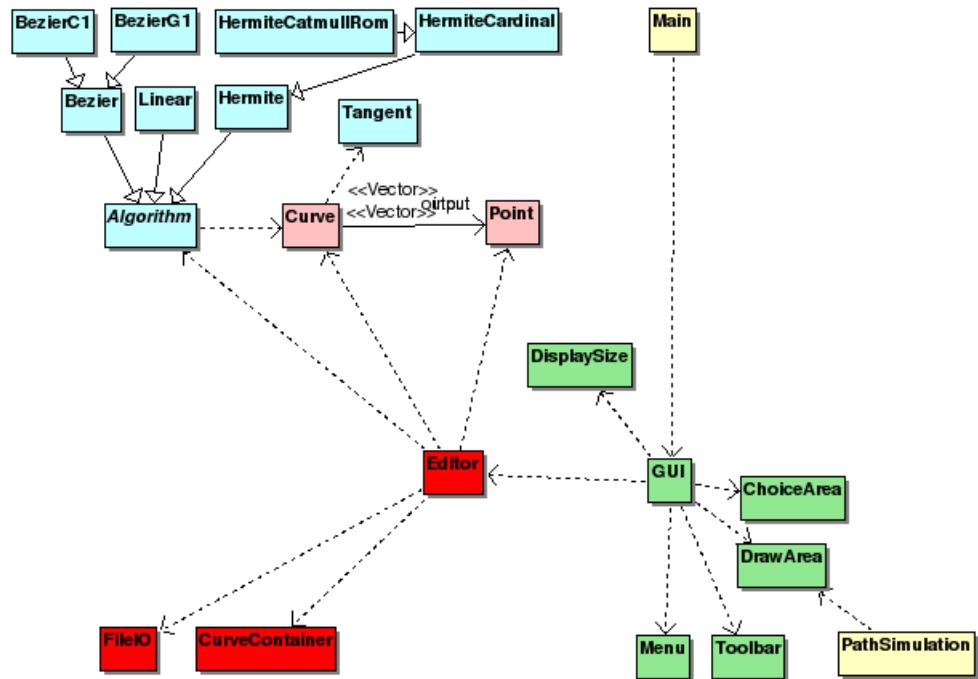


Figuur 12: Hermite Cardinal

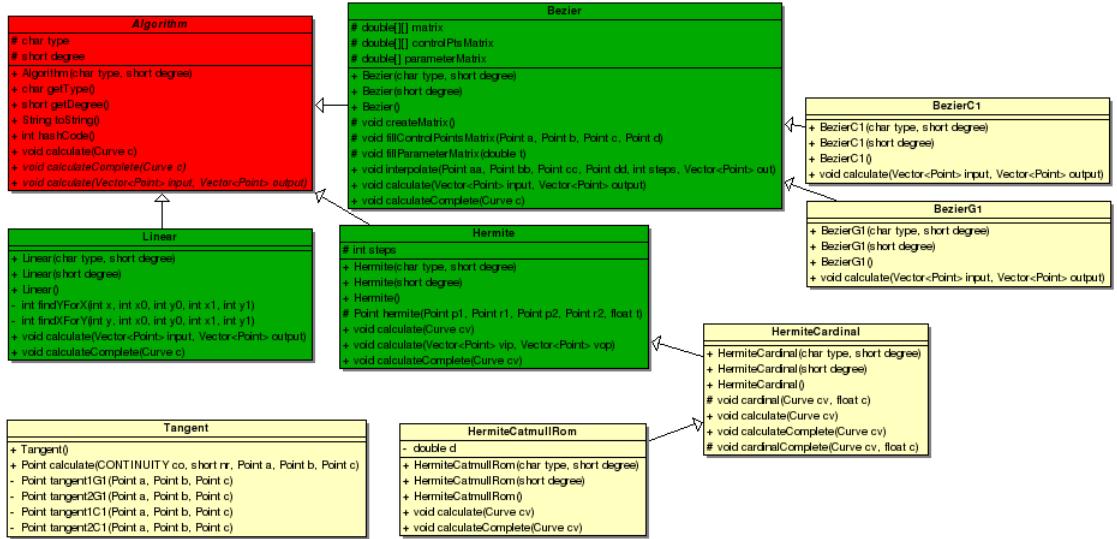


Figuur 13: Hermite Catmull-Rom

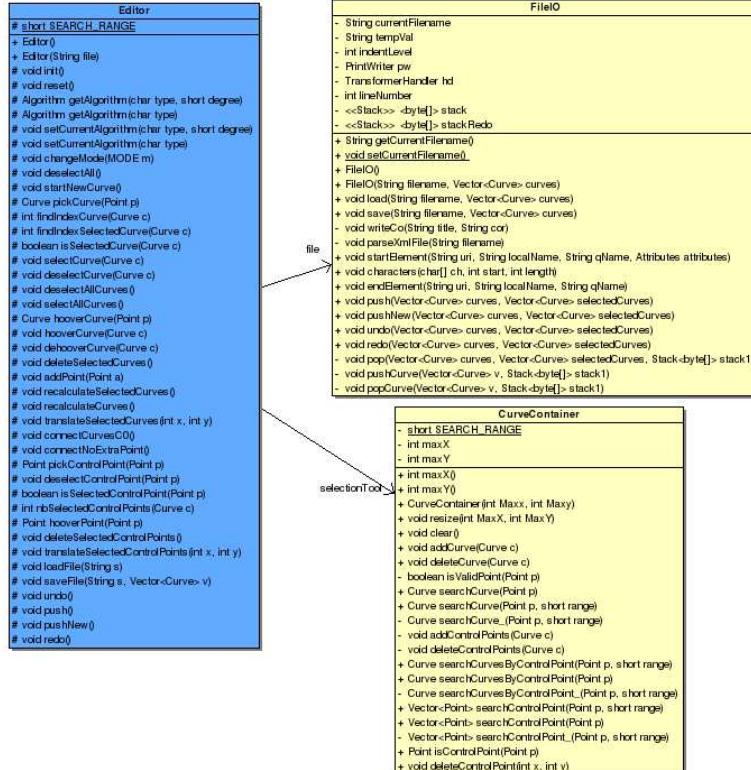
B UML-diagrammen



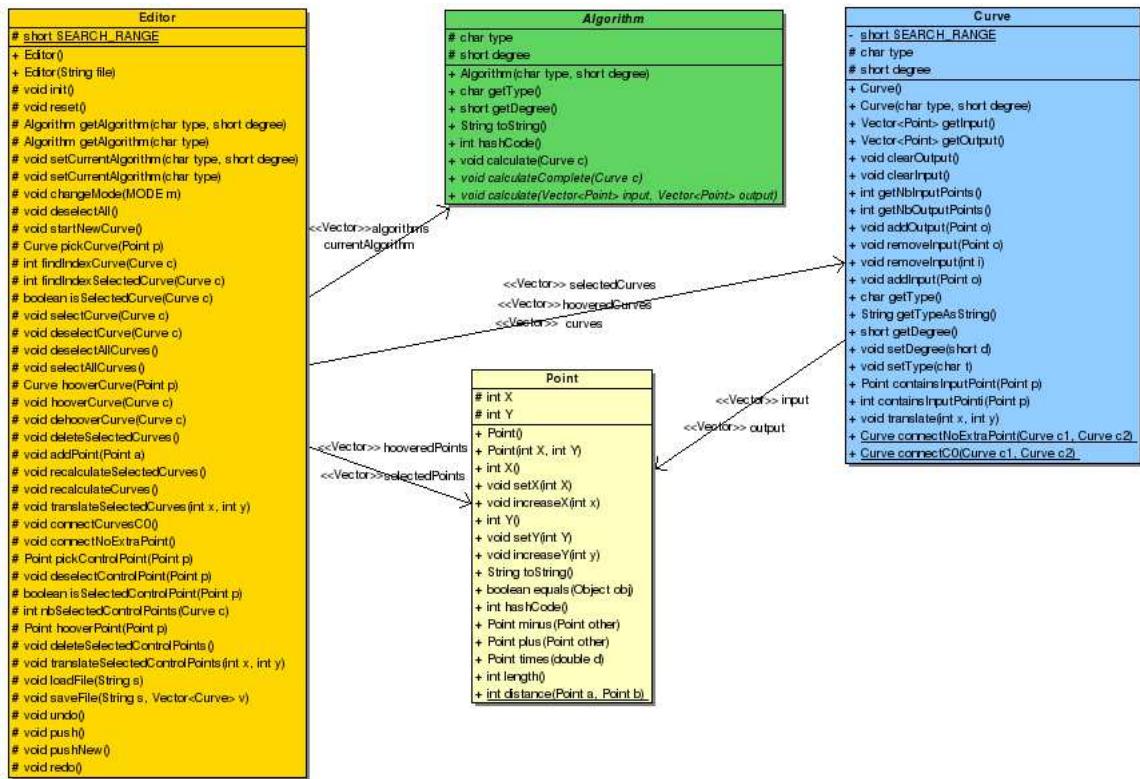
Figuur 14: Totaal diagram



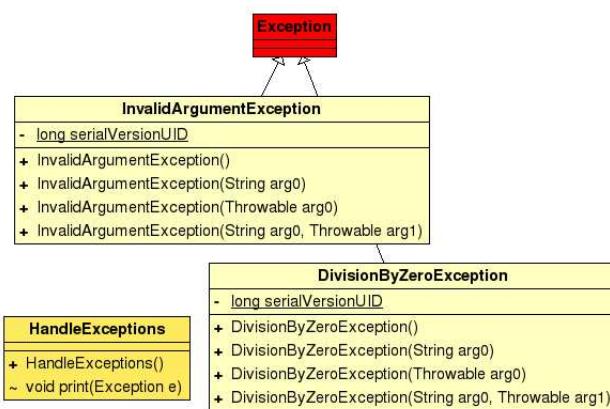
Figuur 15: Algorithm



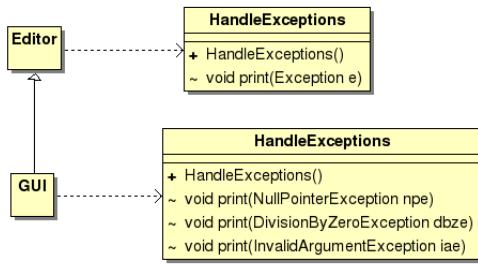
Figuur 16: Core



Figuur 17: Curves (ADT's + interacties daartussen)



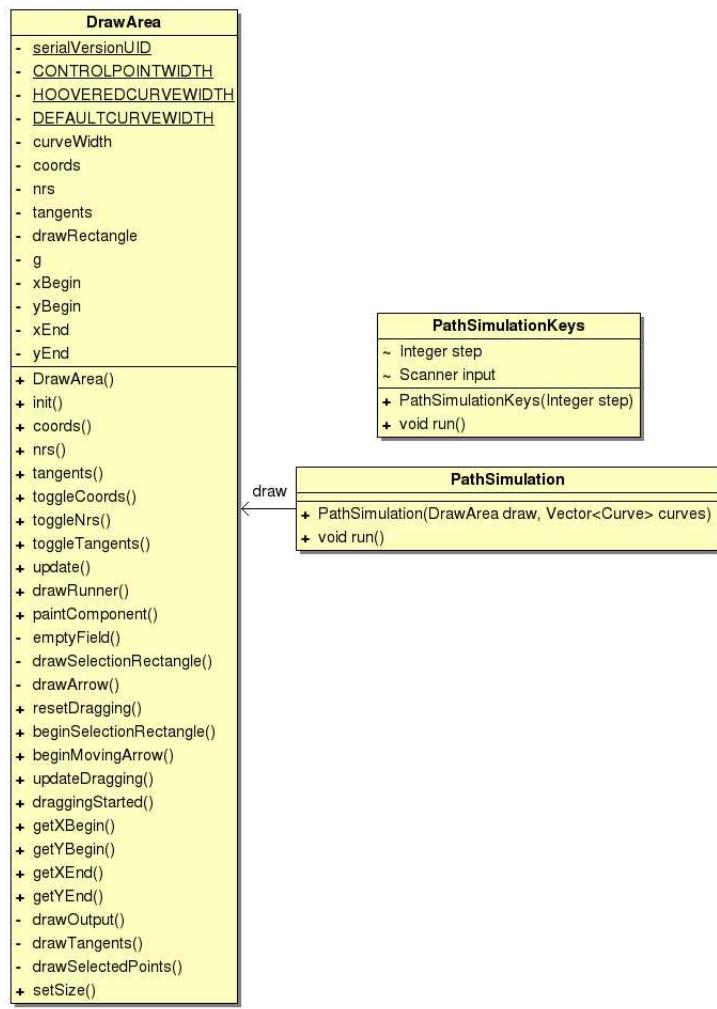
Figuur 18: Exceptions



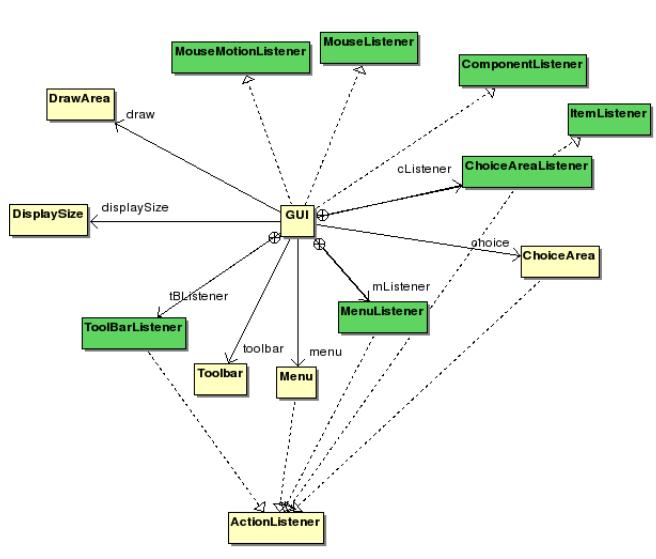
Figuur 19: HandleExceptions



Figuur 20: GUI



Figuur 21: Path Simulation Tool + interactie met DrawArea



Figuur 22: Event Listeners

C Referenties

Prof. dr. F. VAN REETH, Prof. dr. P. BEKAERT, 2007-2008, Computer Graphics

- http://en.wikipedia.org/wiki/Bezier_curve (Bezier)
- <http://home.scarlet.be/piet.verplancken3/bezier/node9.html> (Bézier)
- http://en.wikipedia.org/wiki/Catmull-Rom_spline (Hermite)
- <http://java.sun.com/javase/6/docs/api/> (Java)
- <http://java.sun.com/docs/books/tutorial/uiswing/events/intro.html> (Event en Listeners)
- <http://www.w3schools.com/xml/default.asp> (XML)
- <http://www.w3schools.com/dtd/default.asp> (DTD)
- <http://www.totheriver.com/learn/xml/xmltutorial.html> (XML)
- <http://www.kde.org/> (icoontjes)

D Handleiding

Curve Editor - Documentatie

Inhoud

I. Inleiding

II. Snelle start

1. Curve Editor starten
2. Curve Editor afsluiten
3. Curve Editor-omgeving
4. Canvas inladen
5. Canvas opslaan
6. Canvas清空

III. Gebruik

1. Curves
 - ◊ Nieuwe curve aanmaken
 - ◊ Curves (de)selecteren
 - ◊ Curves verplaatsen
 - ◊ Curves verwijderen
 - ◊ Curves verbinden
2. Controlepunten
 - ◊ Punten (de)selecteren
 - ◊ Punten verplaatsen
 - ◊ Punten verwijderen
 - ◊ Nieuw punt toevoegen
3. Andere
 - ◊ Undo/Redo
 - ◊ Path Simulation Tool
 - ◊ Curveweergave wijzigen

IV. Beschikbare algoritmen

- ◆ Linear
- ◆ Bezier C0
- ◆ Bezier G1
- ◆ Bezier C1
- ◆ Hermite normaal
- ◆ Hermite Cardinal
- ◆ Hermite Catmull-Rom

V. Ontwikkelaars

I. Inleiding

Bedankt om voor Curve Editor te kiezen! Deze handleiding zal u zo goed mogelijk proberen bij te staan met het oplossen van uw problemen met Curve Editor. Moest u echter op een probleem stuiten dat niet beschreven wordt in deze documentatie, dan kan u zich steeds beroepen op de hulp van onze medewerkers. Deze zijn te contacteren op:

- ◆ CE Straat 123
- ◆ CE City 897
- ◆ CE Country
- ◆ #\$/#/\$.#\$.#\$

II. Snelle start

Curve Editor - Documentatie - Inhoud

1. Curve Editor starten

U heeft de volgende mogelijkheden om het programma te starten:

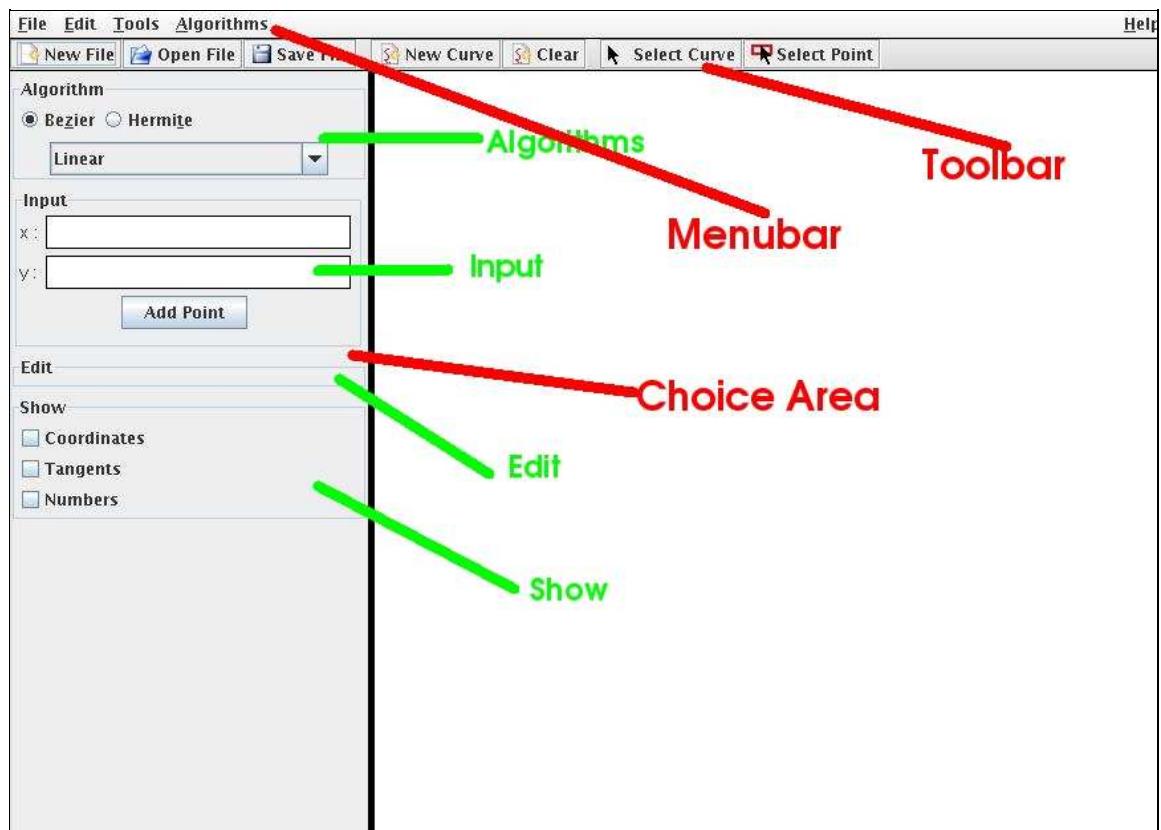
- ◊ Dubbelklik op het bestand CurveEditor.jar
- ◊ Of voer uit vanuit de command line: java -jar CurveEditor.jar

2. Curve Editor afsluiten

U heeft de volgende mogelijkheden om het programma te beëindigen:

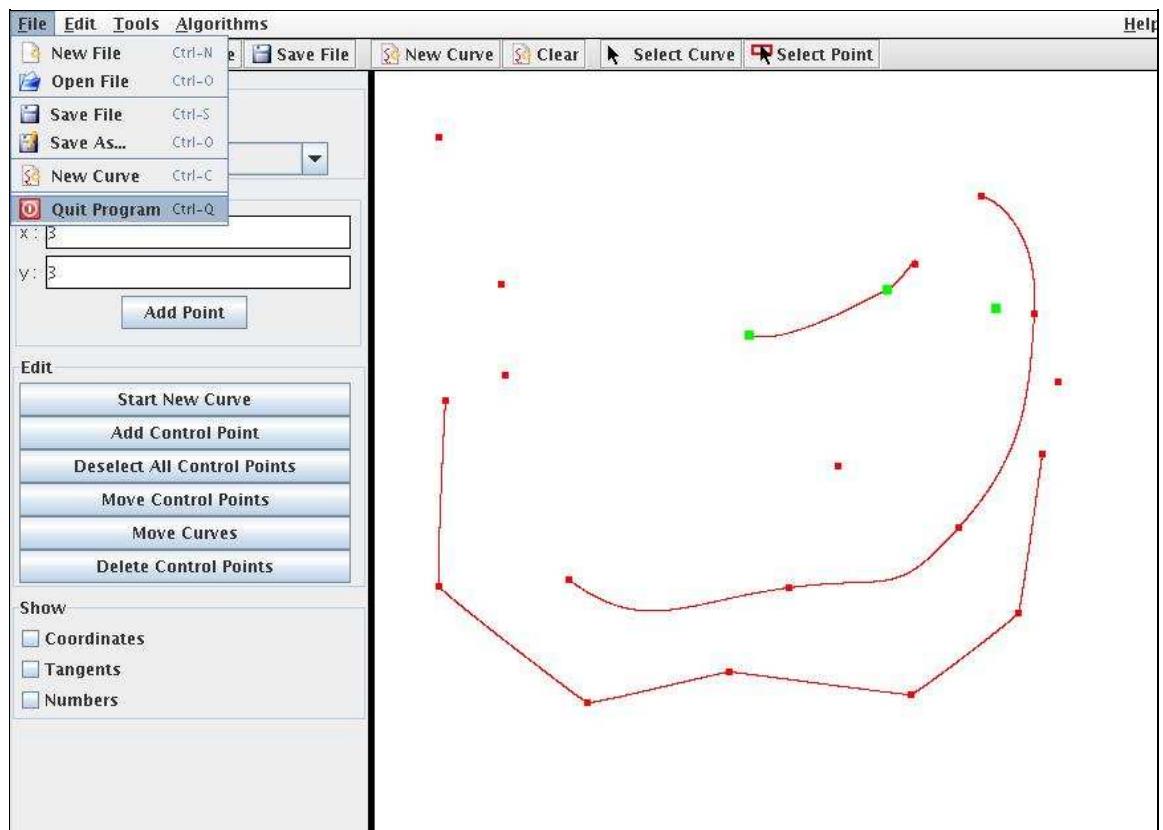
- ◊ op het kruisje rechtsboven in het applicatievenster klikken
- ◊ in de menubalk: File –> Quit Program
- ◊ gebruik de toetsencombinatie Ctrl-Q

3. Curve Editor-omgeving

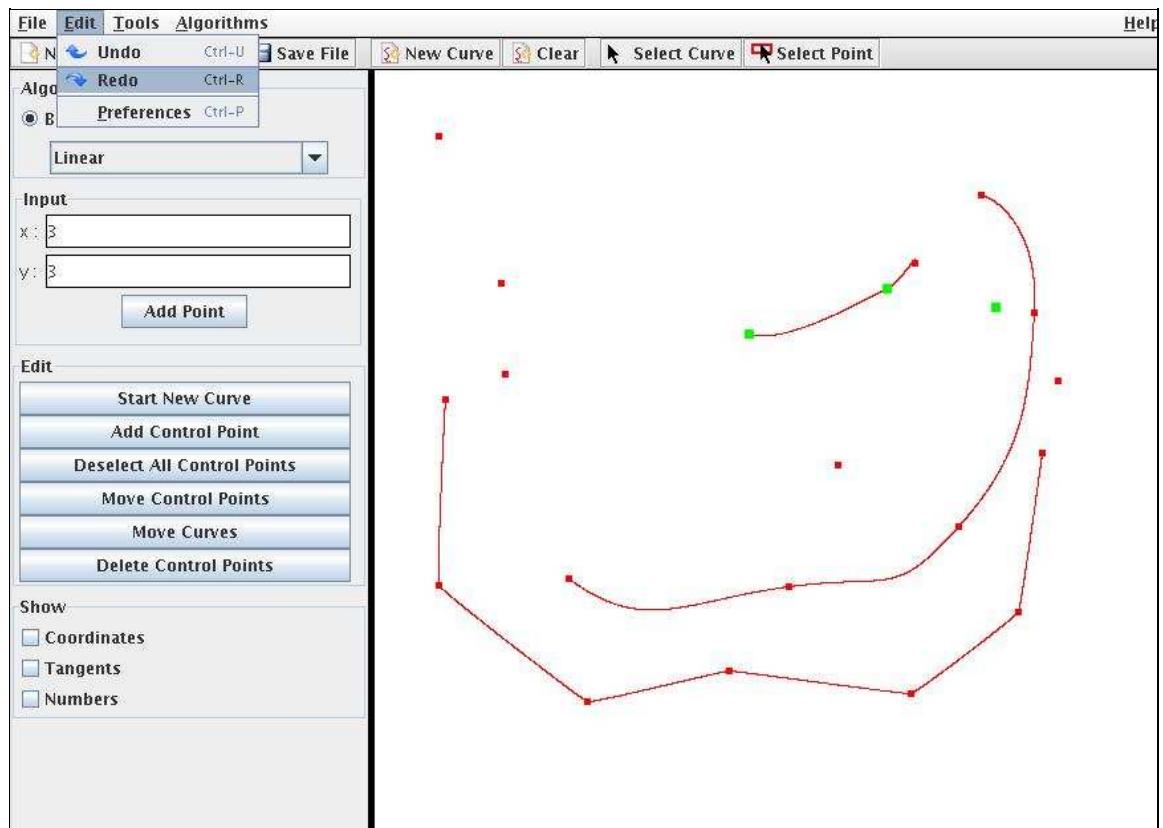


Figuur 1: Curve Editor: werk-omgeving

Curve Editor - Documentatie - Inhoud

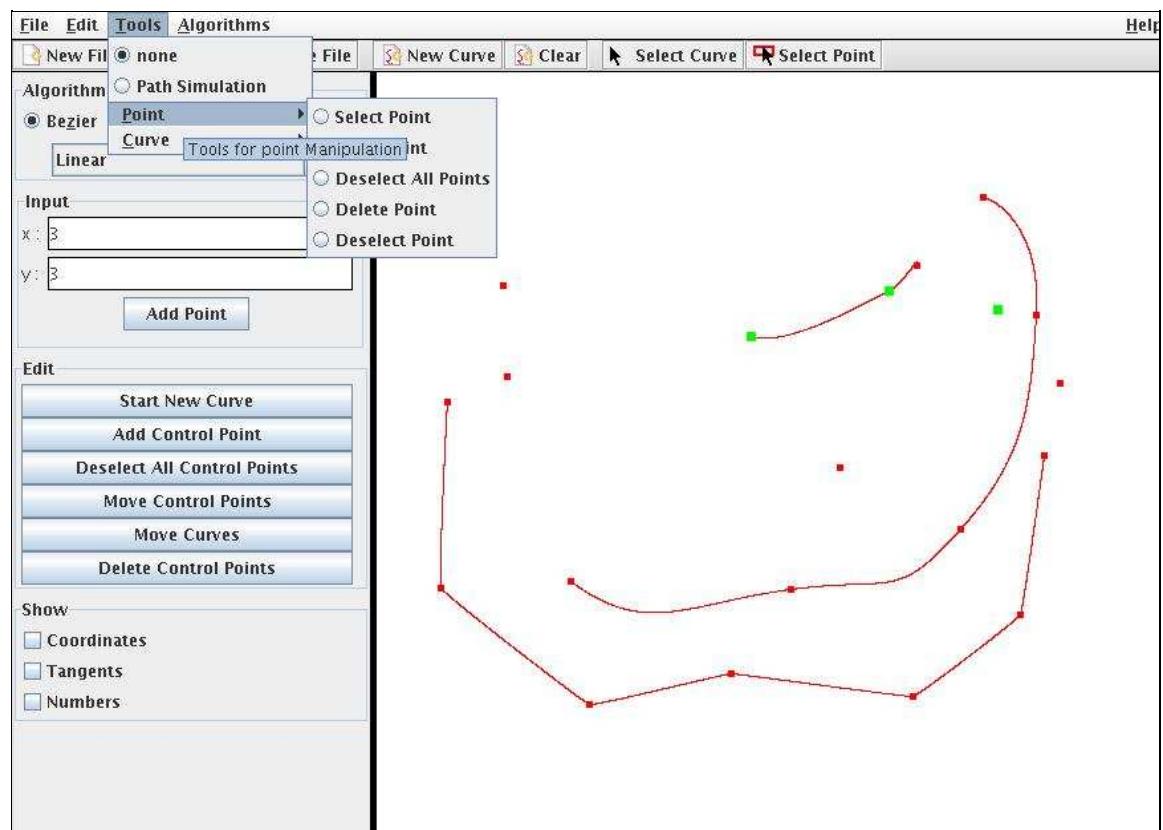


Figuur 2: Curve Editor: Menu, File

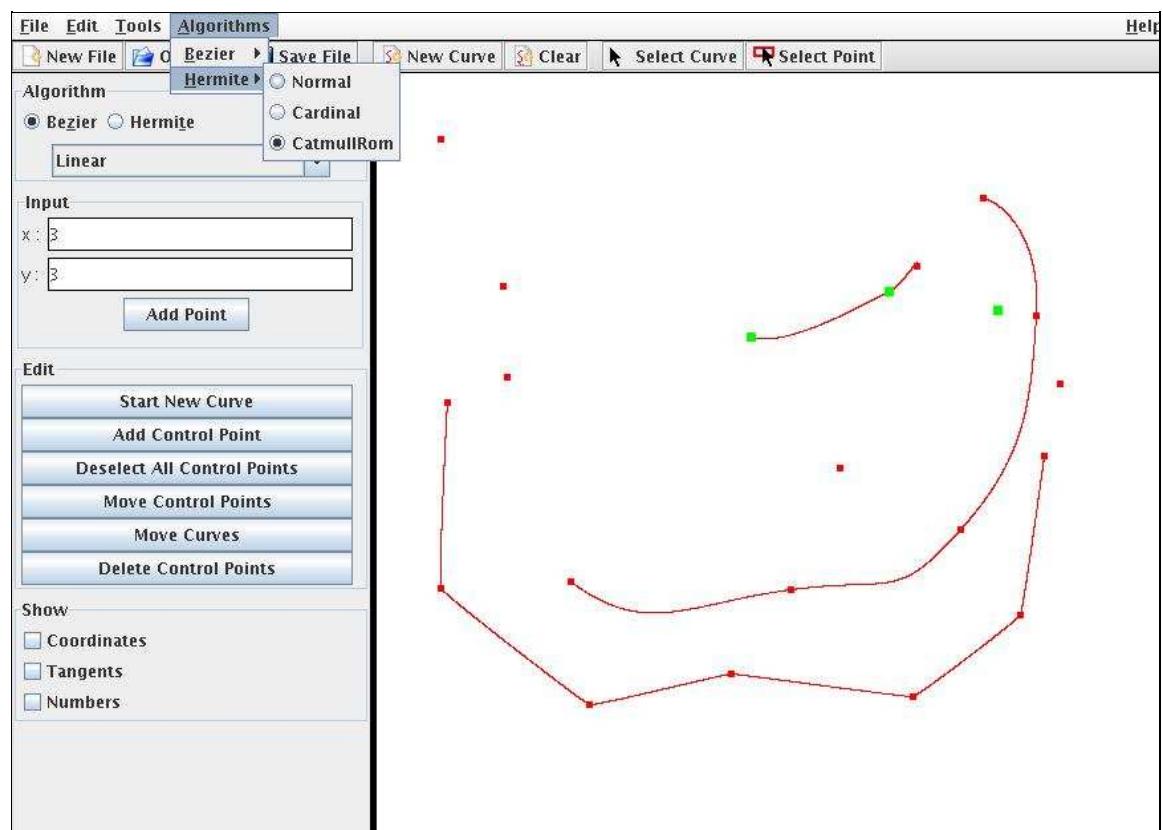


Figuur 3: Curve Editor: Menu, Edit

Curve Editor - Documentatie - Inhoud

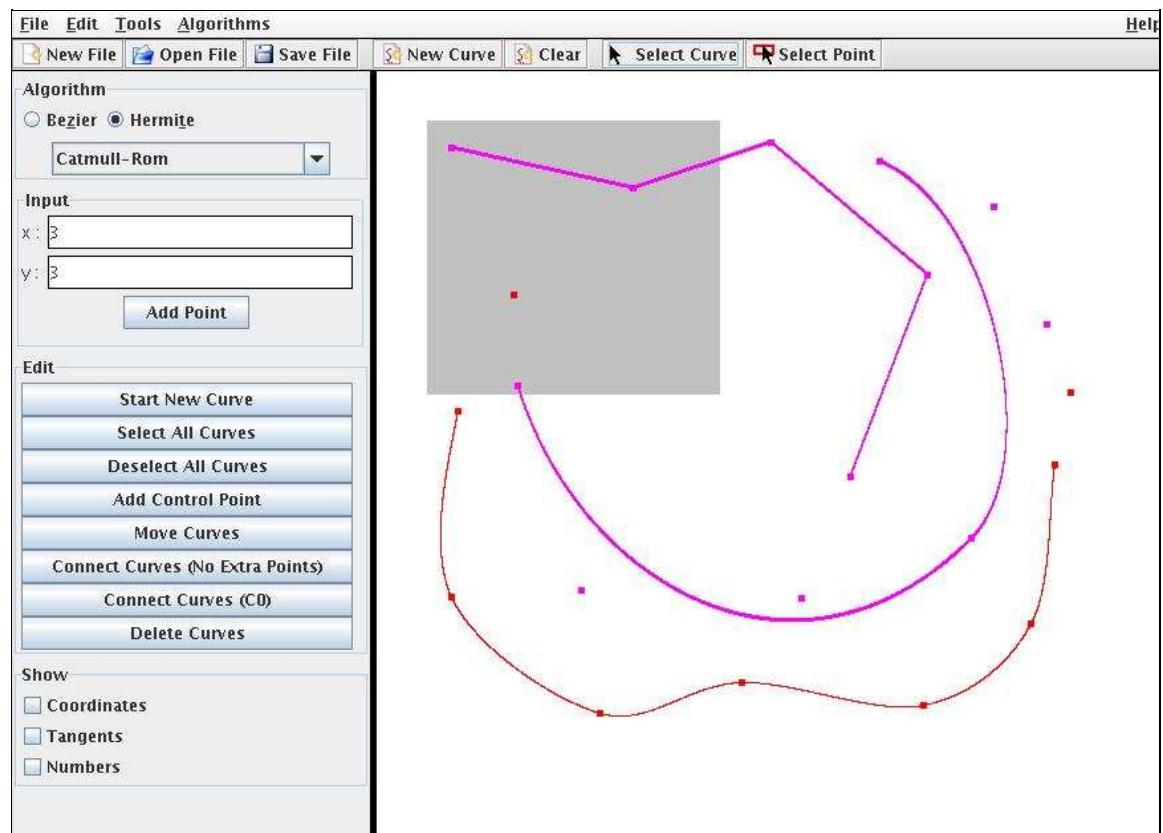


Figuur 4: Curve Editor: Menu, Tools



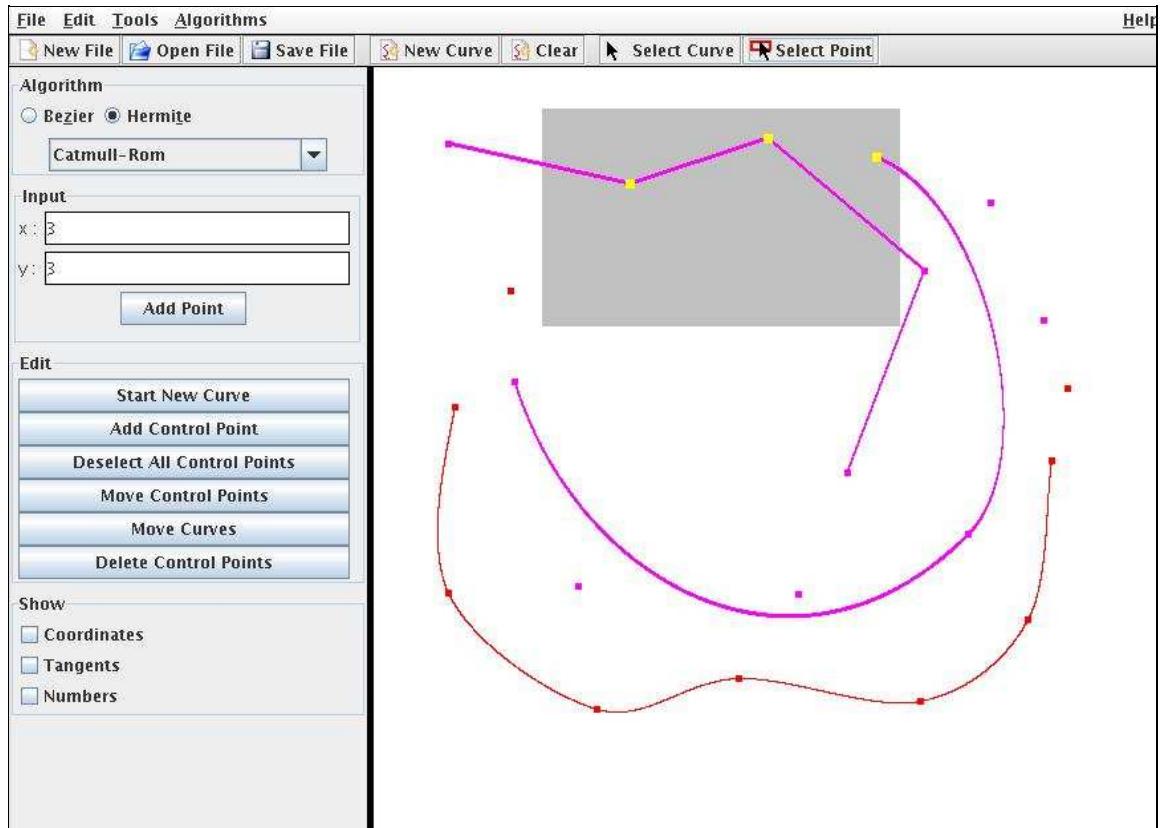
Curve Editor - Documentatie - Inhoud

Figuur 5: Curve Editor: Menu, Algorithms



Figuur 6: Curve Editor: Choice Area, Edit, Curves

Curve Editor - Documentatie - Inhoud



Figuur 7: Curve Editor: Choice Area, Edit, Points

De Curve Editor bestaat uit een menubalk (zie figuur 1) waar u de mogelijkheid wordt gegeven van allerhande taken uit te voeren en instellingen te veranderen. Kort geschatst zijn volgende acties vanuit de menubalk mogelijk:

- ◊ File –> (zie figuur 2)
 - New File (cleart het canvas en begint met een nieuw bestand)
 - Open File (laadt een bestand in, zonder het canvas te clearen)
 - Save File (slaat het canvas op, in een *.xml-bestand, of in een *.png/gif-bestand)
 - Save As... (slaat het canvas op in het vorig gebruikte bestand, indien dat bestaat)
 - New Curve (deselecteert alle curves en begint een nieuwe curve op het canvas)
 - Quit Program (sluit het programma af)
- ◊ Edit –> (zie figuur 3)
 - Undo (doet de laatste verandering teniet)
 - Redo (doet de laatste teniet gedane verandering opnieuw)
 - Preferences (geeft de mogelijkheid van de lijndiktes e.d. in te stellen)
- ◊ Tools –> (zie figuur 4)
 - None
 - Path Simulation (start de wegsimulatielool)
 - Point (submenu geeft de mogelijkheid bewerkingen te doen op de controlepunten ((de)selecteren, verplaatsen, deleten, ...))
 - Curve (submenu geeft de mogelijkheid bewerkingen te doen op de curves ((de)selecteren, verplaatsen, deleten, verbinden, ...))
- ◊ Algorithms –> (zie figuur 5)

Curve Editor - Documentatie - Inhoud

- Bezier (submenu laat u kiezen uit de verschillende Bezier-varianten)
- Hermite (submenu laat u kiezen uit de verschillende Hermite-varianten)

◊ Help ->

- Quick Howto's (laat deze sectie inladen)
- Documentation (laat dit document inladen)
- About

Voorts kunnen deze commando's ook met de volgende sneltoetscombinaties aangeroepen worden:

- ◊ Ctrl-Q: de applicatie afsluiten
- ◊ Ctrl-O: een bestand openen
- ◊ Ctrl-S: naar een bestand opslaan
- ◊ Ctrl-N: een nieuw bestand beginnen
- ◊ Ctrl-C: een nieuwe curve starten
- ◊ Ctrl-U: de laatst gedane actie ongedaan maken
- ◊ Ctrl-R: de laatst ongedaan gemaakte actie terug gedaan maken
- ◊ Ctrl-P: de instellingsmogelijkheden weergeven

Verder vindt u het volgende terug in de Curve Editor-omgeving (zie figuur 1):

◊ Toolbar:

- New File: een nieuw bestand beginnen
- Open File: een bestand openen
- Save File: naar een bestand opslaan
- New Curve: een nieuwe curve beginnen
- Clear: alle curves en punten verwijderen
- Select Curve: een curve selecteren (nadien kan men dan curves in het canvas selecteren)
- Select Point: een controlepunt selecteren (nadien kan men dan controlepunten in het canvas selecteren)

◊ Choice Area:

- Algorithms: na selectie van Bezier of Hermite kan men in het lijstje het gewenste algoritme kiezen. Nieuwe curves zullen dan met dit algoritme berekend worden, alsook de huidige geselecteerde curves.
- Input: dit geeft de mogelijkheid om een nieuw controlepunt toe te voegen a.h.v. exacte coördinaten. U vult de gewenste getallen in en klikt op "Add Point"; het punt zal dan aan de geselecteerde curves toegevoegd worden en weergegeven worden.
- Show: het aanvinken van een keuze zorgt ervoor dat deze in het canvas wordt weergegeven.
- Edit: dit kan twee verschillende vormen aannemen, afhangende van het feit of u curves of controlepunten geselecteerd hebt.

1. Curves (zie figuur 6):

- ◆ Start New Curve: een nieuwe curve starten
- ◆ Select All Curves: alle curves selecteren
- ◆ Deselect All Curves: alle curves deselecteren
- ◆ Add Control Point: een nieuw controlepunt aan de geselecteerde curves toevoegen
- ◆ Move Curves: de geselecteerde curves verplaatsen (hierna kan men door de cursor in het canvas te draggen de curves verplaatsen)

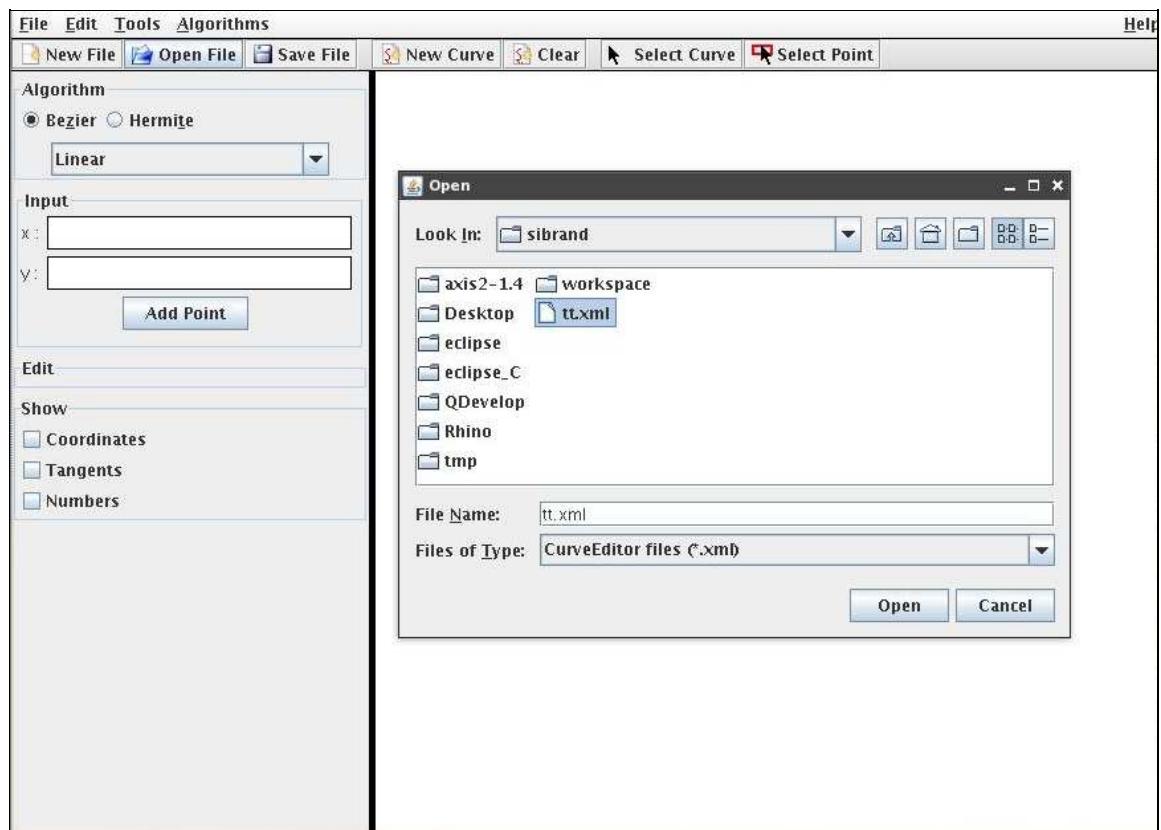
Curve Editor - Documentatie - Inhoud

- ◆ Connect Curves (No Extra Points): de geselecteerde curves verbinden en op hun oorspronkelijke plaats laten
- ◆ Connect Curves (C0): de geselecteerde curves verbinden en verplaatsen naar het einde van de vorige curve
- ◆ Delete Curves: alle geselecteerde curves verwijderen

2. Controlepunten (zie figuur 7):

- ◆ Start New Curve: een nieuwe curve starten
- ◆ Add Control Point: een nieuw controlepunt aan de geselecteerde curves toevoegen
- ◆ Deselect All Control Points: alle controlepunten deselecteren
- ◆ Move Control Points: de geselecteerde punten verplaatsen (hierna kan men door de cursor in het canvas te dragen de punten verplaatsen)
- ◆ Move Curves: de geselecteerde curves verplaatsen (hierna kan men door de cursor in het canvas te dragen de curves verplaatsen)
- ◆ Delete Control Points: alle geselecteerde punten verwijderen

4. Canvas inladen



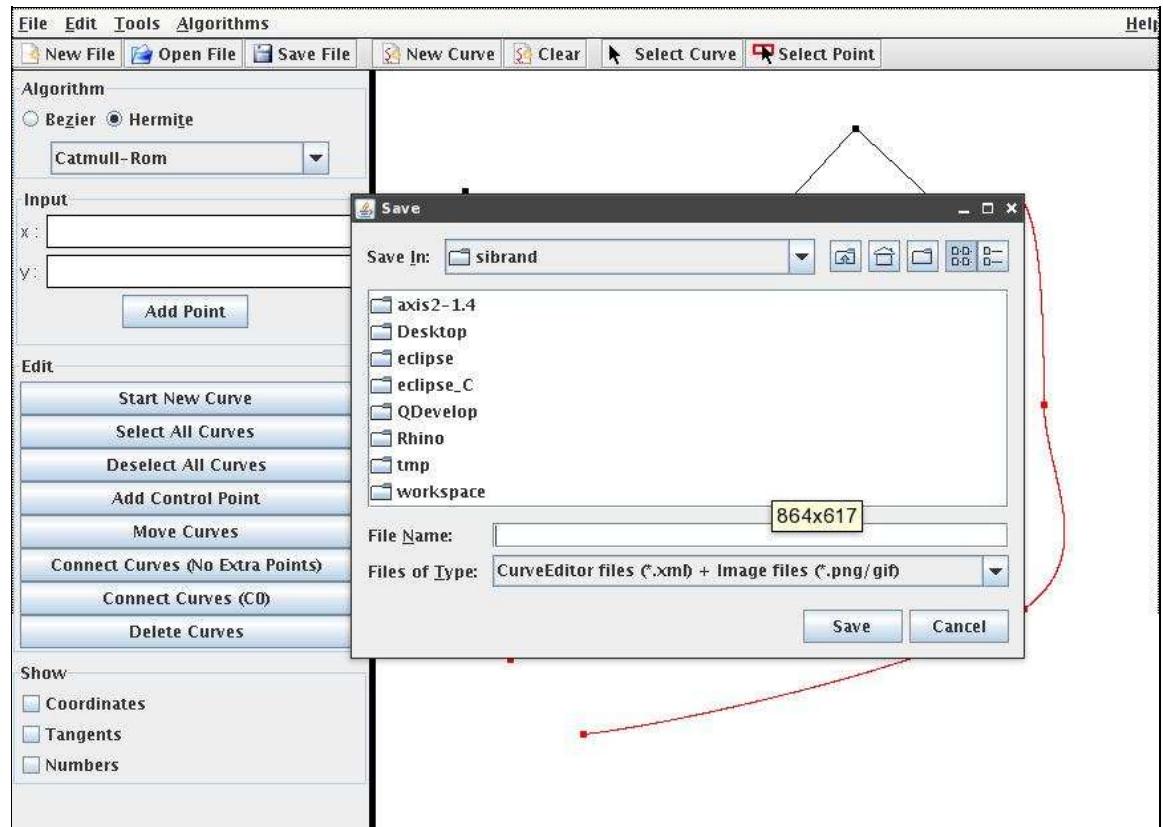
Figuur 8: Curve Editor: canvas/curve inladen

Een bestand inladen gebeurt door vanuit de menubalk "Open File" te kiezen of de sneltoetscombinatie Ctrl-O te gebruiken. Na dit gedaan te hebben, krijgt men het venster dat in figuur 8 te zien is. In dit venster navigeert men dan naar het gewenste *.xml-bestand; na selectie van dat bestand klikt men op "Open" om het bestand in te laden. Inladen van

Curve Editor - Documentatie - Inhoud

bestanden zorgt niet voor het clearen van het canvas; op deze manier is het mogelijk curves uit meerdere bestanden tegelijkertijd te gebruiken: men laadt dan gewoon de nodige bestanden in.

5. Canvas opslaan



Figuur 9: Curve Editor: canvas/curves opslaan

Een nieuw bestand aanmaken gebeurt door vanuit de menubalk "Save As" te kiezen of de sneltoetscombinatie Ctrl-S te gebruiken. Om het vorig weggeschreven bestand te overschrijven met de huidige inhoud van het canvas, dan kiest men vanuit de menubalk "Save File". Na een van de twee acties gedaan te hebben, krijgt men het venster dat in figuur 9 te zien is. In dit venster navigeert men dan naar het gewenste bestand dat men wilt overschrijven; na selectie van dat bestand klikt men op "Save" om het bestand weg te schrijven. Men kan i.p.v. een reeds bestaand bestand te selecteren ook zelf een bestandsnaam in het tekstveld "File Name" ingeven: dit schrijft het canvas weg naar dat bestand. Men kan kiezen om het canvas in een *.xml-bestand op te slaan; dat geeft de mogelijkheid om het daarna terug in het programma in te laden en verder te gaan. Men kan ook kiezen om het canvas als een afbeelding in een *.png- of *.gif-bestand op te slaan. Indien een ander bestandsformaat gekozen wordt, dan zal het uiteindelijke bestand de *.xml-informatie bevatten.

6. Canvas clearen

Het canvas clearen heeft als gevolg dat alle aangemaakte curves verwijderd worden en men terug van een lege canvas vertrekt. Dit kan op de volgende manieren bekomen worden:

- ◊ Menu –> File –> New File: een nieuw bestand beginnen heeft hetzelfde effect als alle curves verwijderen

Curve Editor - Documentatie - Inhoud

- ◊ Toolbar –> New File: een nieuw bestand beginnen heeft hetzelfde effect als alle curves verwijderen
- ◊ Toolbar –> Clear

III. Gebruik

◆ Nieuwe Curve Aanmaken

Volgende mogelijkheden zijn beschikbaar om aan een nieuwe curve te beginnen:

- ◊ Gebruik de sneltoetsen Ctrl-C
- ◊ Menu –> File –> New Curve
- ◊ Toolbar –> New Curve
- ◊ Indien in Choice Area het Edit-deel actief is: klik op "Start New Curve"

Nadat u één van de vorige acties gedaan hebt, kunt u in het canvas aan een nieuwe curve beginnen door een eerste nieuw controlepunt aan te klikken of in te geven.

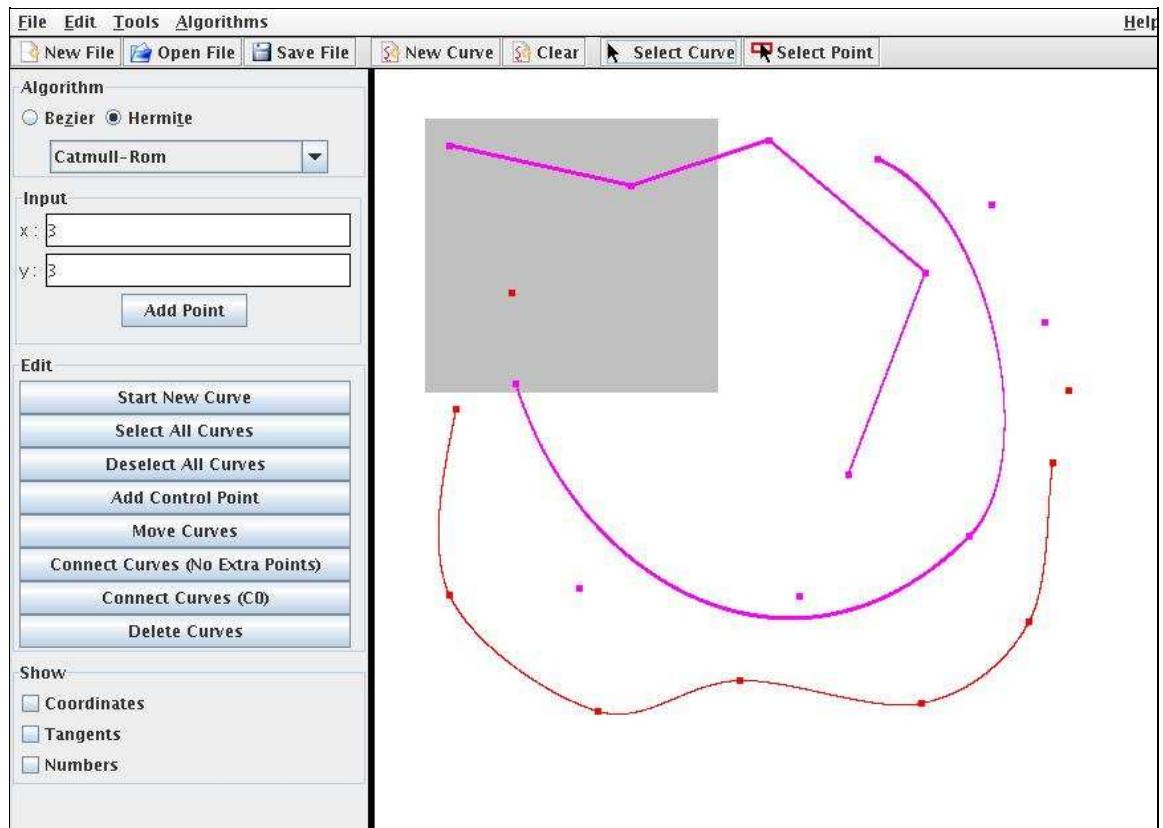
◆ Curves (de)selecteren

Toolbar –> Select Curve zorgt ervoor dat men curves kan (de)selecteren, als dat al niet het geval was. Reeds geselecteerde curves blijven geselecteerd.

Om curves te selecteren kan men de muiscursor dragen: men duwt de linkermuisknop in, sleept de cursor waardoor een selectierechthoek zichtbaar wordt, en men laat de linkermuisknop terug los. Alle curves die punten bevatten in dat selectierechthoekje zijn nu geselecteerd. Een andere manier is om curves aan te klikken: dit zorgt er ook voor dat geselecteerde curves gedeselecteerd kunnen worden. Nog een andere mogelijkheid is om ALLE aangemaakte curves te selecteren door op Choice Area –> Edit –> Select All Curves te klikken. ALLE geselecteerde curves worden gedeselecteerd door op Choice Area –> Edit –> Deselect All Curves te klikken.

Geselecteerde curves worden in het rood weergegeven, niet-geselecteerde in het zwart (zie figuur 10). Curves die na het loslaten van de linkermuisknop of curves die na klikken op de huidige positie geselecteerd zullen worden, worden in magenta weergegeven (zie figuur 10).

Curve Editor - Documentatie - Inhoud



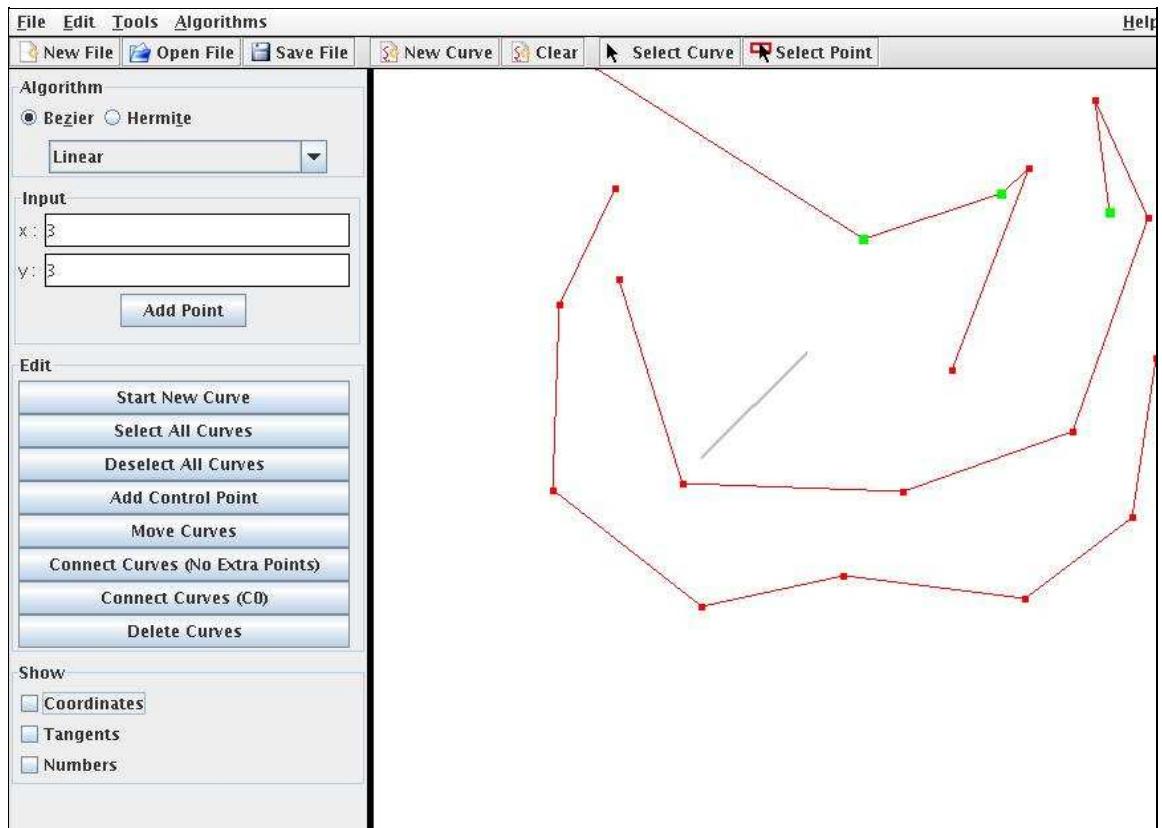
Figuur 10: Curve Editor: curves selecteren

◆ Curves verplaatsen

Dit is enkel mogelijk indien men curves (of controlepunten met de daaraan verbonden curves) geselecteerd heeft.

Choice Area –> Edit –> Move Curves activeert de move-functionaliteit. Die werkt als volgt: men houdt de linkermuisknop ingeduwd op een plaats in het canvas, dit is de startpositie. Nu kan men de cursor over het canvas slepen en alle curves worden verplaatst over een afstand die gelijk is aan de afstand tussen de startpositie van de cursor en de huidige positie van de cursor. Die afstand wordt weergegeven door een grijze lijn (zie figuur 11). Laat men de linkermuisknop los, dan zijn de curves definitief verplaatst.

Curve Editor - Documentatie - Inhoud



Figuur 11: Curve Editor: curves verplaatsen

♦ Curves verwijderen

Men kan uiteraard kiezen om ALLE curves te verwijderen door heel het canvas te clearen, maar men kan ook enkel de geselecteerde curves verwijderen. Dit doet men door op Choice Area –> Edit –> Delete Curves te klikken. Dit zorgt ervoor dat het canvas enkel nog maar bestaat uit de niet-geselecteerde curves.

♦ Curves verbinden

Indien er curves geselecteerd zijn, dan kan men ervoor kiezen om deze tot één lange curve te verbinden. Ongeacht de gekozen manier wordt de curve berekend a.h.v. het algoritme van de eerst geselecteerde curve. Vervolgens wordt elke volgende geselecteerde curve verbonden aan de lange curve om uiteindelijk tot een grote curve te komen.

Er zijn twee verbindingsmanieren:

◊ Choice Area –> Edit –> Connect Curves (No Extra Points)

De nieuw aangemaakte curve krijgt de eigenschappen van de eerst geselecteerde curve en AL de controlepunten van de geselecteerde curves. Het eerste controlepunt van een geselecteerde curve volgt op het laatste controlepunt van de vorige geselecteerde curve. De controlepunten behouden hun oorspronkelijke posities met andere woorden.

◊ Choice Area –> Edit –> Connect Curves (C0)

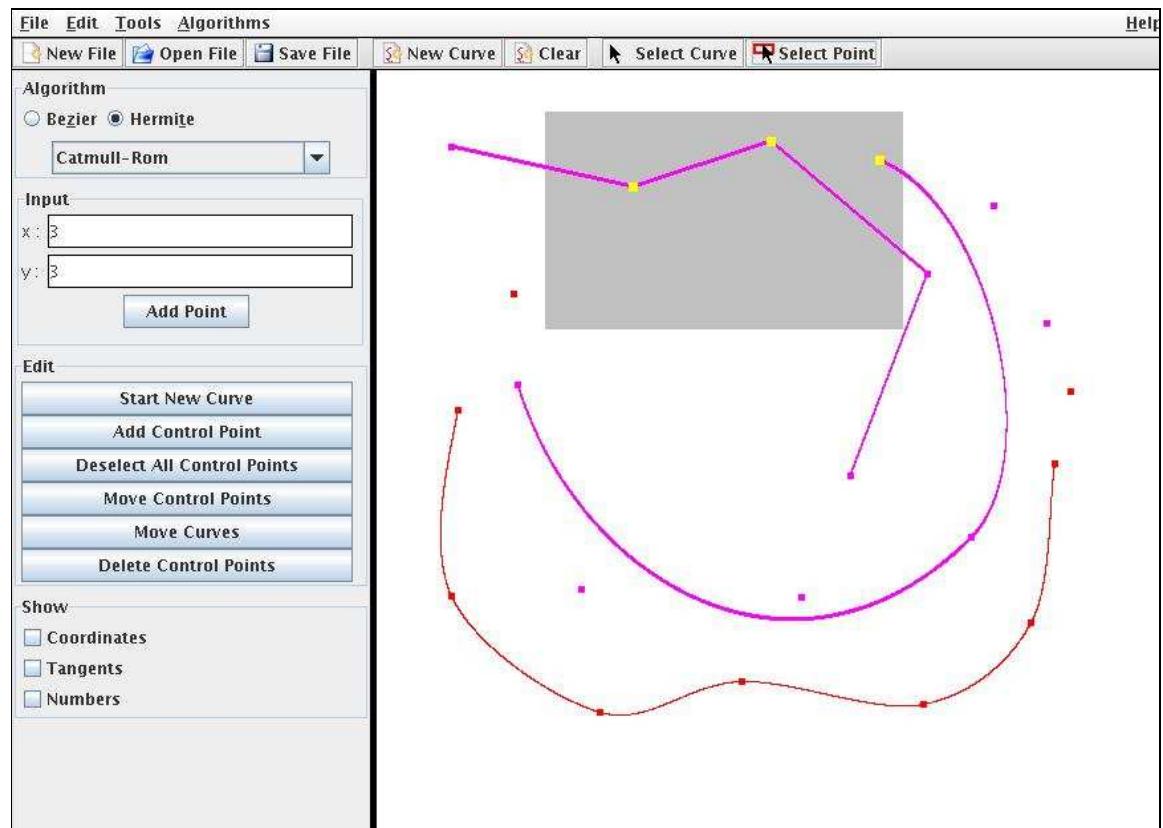
De nieuw aangemaakte curve krijgt de eigenschappen van de eerst geselecteerde curve en AL de controlepunten van de geselecteerde curves. Het eerste controlepunt van een geselecteerde curve valt samen met het laatste controlepunt van de vorige geselecteerde curve. De controlepunten van die volgende te verbinden curve worden verschoven over een afstand zodanig dat die twee punten samenvallen.

◆ Punten (de)selecteren

Toolbar –> Select Point zorgt ervoor dat men controlepunten kan (de)selecteren, als dat al niet het geval was. Reeds geselecteerde controlepunten blijven geselecteerd.

Om controlepunten te selecteren kan men de muiscursor draggen: men duwt de linkermuisknop in, sleept de cursor waardoor een selectierechthoek zichtbaar wordt, en men laat de linkermuisknop terug los. Alle controlepunten in dat selectierechthoek zijn nu geselecteerd. Een andere manier is om controlepunten aan te klikken: dit zorgt er ook voor dat geselecteerde controlepunten gedeselecteerd kunnen worden. ALLE geselecteerde controlepunten worden gedeselecteerd door op Choice Area –> Edit –> Deselect All Control Points te klikken.

Geselecteerde controlepunten worden in het groen weergegeven, niet-geselecteerde in het zwart of rood, naargelang de selectiestatus van de curve waartoe zij behoren (zie figuur 12). Controlepunten die na het loslaten van de linkermuisknop of controlepunten die na klikken op de huidige positie ge(de)selecteerd zullen worden, worden in geel weergegeven (zie figuur 12). Geselecteerde of gehoverde punten zorgen er ook voor dat de curves waarvan dat punt een controlepunt is, ook geselecteerd of gehoverd zijn.



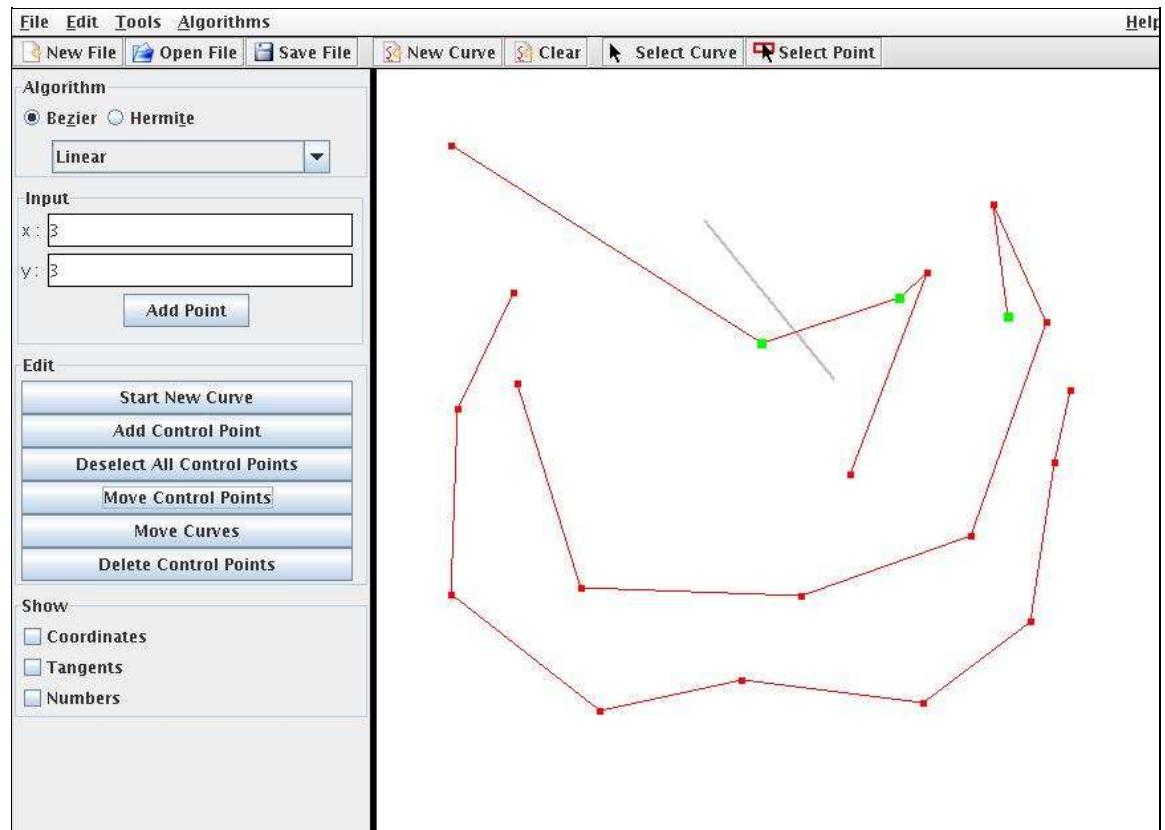
Figuur 12: Curve Editor: controlepunten selecteren

◆ Punten verplaatsen

Dit is enkel mogelijk indien men controlepunten geselecteerd heeft, uiteraard Choice Area –> Edit –> Move Control Points activeert de move-functionaliteit. Die werkt als volgt: men houdt de linkermuisknop ingeduwd op een plaats in het canvas, dit is de startpositie. Nu kan men de cursor over het canvas slepen en alle geselecteerde

Curve Editor - Documentatie - Inhoud

controlepunten worden verplaatst over een afstand die gelijk is aan de afstand tussen de startpositie van de cursor en de huidige positie van de cursor. Die afstand wordt weergegeven door een grijze lijn (zie figuur 13). Laat men de linkermuisknop los, dan zijn de controlepunten definitief verplaatst.



Figuur 13: Curve Editor: punten verplaatsen

◆ Punten verwijderen

Dit doet men door op Choice Area –> Edit –> Delete All Control Points te klikken. Dit zorgt ervoor dat de geselecteerde controlepunten uit alle curves tot waartoe zij behoren verwijderd worden.

◆ Nieuw punt toevoegen

Dit is enkel mogelijk indien men curves geselecteerd heeft. Curves zijn geselecteerd als ten minste één controlepunt van de curve geselecteerd is, of wanneer de curve zelf geselecteerd is. Indien men een nieuwe curve aangemaakt heeft, is die automatisch geselecteerd. Een nieuw controlepunt wordt toegevoegd aan alle curves die op dat moment geselecteerd zijn. Choice Area –> Input geeft de mogelijkheid om een controlepunt op een exacte positie toe te voegen. Men vult dan de coördinaten in en klikt op "Add Point". De andere manier is door de add-functionaliteit te activeren, als dat al niet geval was. Dit doet men door op Choice Area –> Edit –> Add Control Point te klikken. Nu kan men in het canvas klikken op een plaats waar men een nieuw controlepunt wilt toevoegen. Men kan punten blijven toevoegen totdat de add-functionaliteit niet meer actief is (wanneer bvb. de move- of select-functionaliteit geactiveerd is).

◆ Undo/Redo

Bepaalde veranderingen aan het canvas kan men ongedaan maken: punten toevoegen, curves verplaatsen, curves verwijderen, ..., kortom: acties die de inhoud van curves echt veranderd hebben. Veranderingen van interpolatiealgoritmen kunnen bvb. niet ongedaan gemaakt worden, omdat deze makkelijk terug kunnen gezet worden via de daartoe bestemde keuzemenu's. Dingen ongedaan maken kan men doen door gebruik te maken van de sneltoetscombinatie Ctrl-U of Menu –> Edit –> Undo.

Indien men een ongedane verandering terug wilt gedaan maken, dan kan men gebruik maken van de sneltoetscombinatie Ctrl-R of Menu –> Edit –> Redo.

◆ Path Simulation Tool

Curve Editor heeft ook een tooltje aan boord: Path Simulation Tool dat men kan activeren via Menu –> Tools –> Path Simulation.

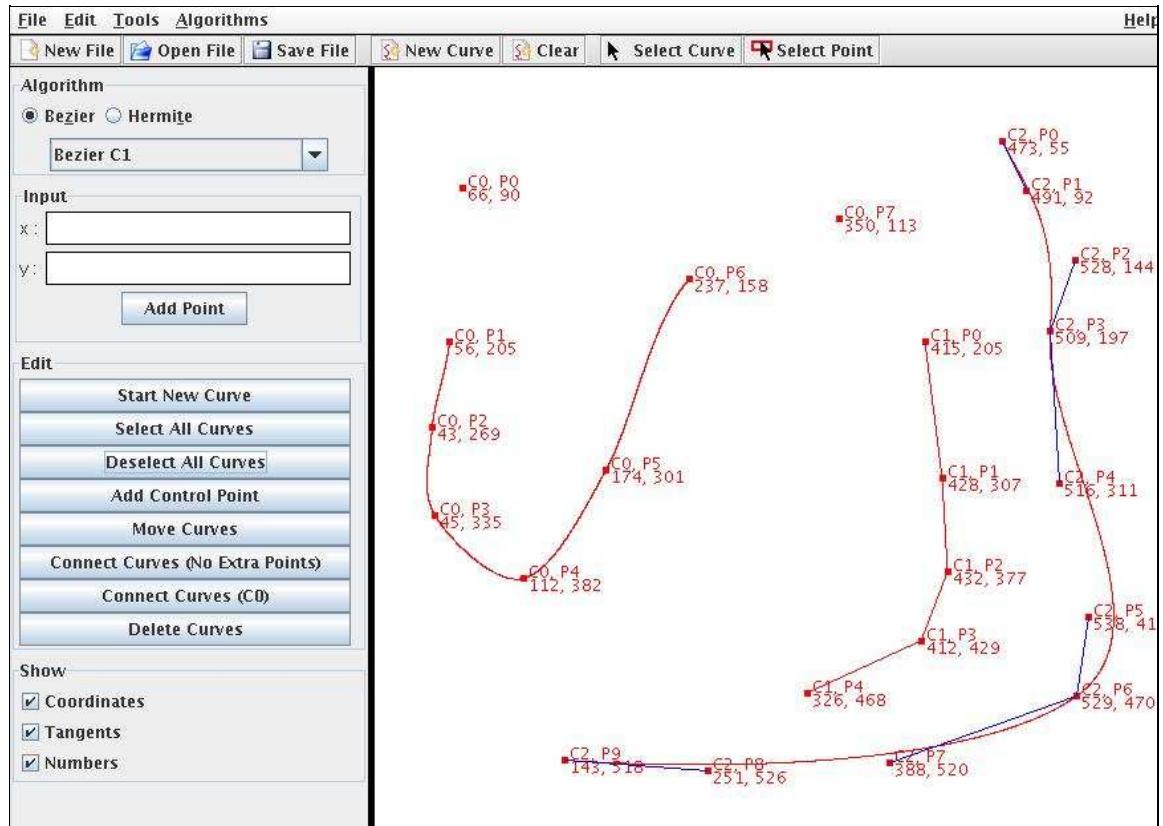
Deze tool laat een bolletje zien dat zich over de eerst geselecteerde curve verplaatst, om zich daarna over de tweede geselecteerde curve te verplaatsen, enzovoort. Het bolletje vertrekt telkens van het begin van de curve om te eindigen in het einde, en dit voor elke geselecteerde curve. De volgorde van curves wordt bepaald door het tijdstip van hun selectie; zo zal de eerst geselecteerde curve eerst gebruikt worden.

◆ Curveweergave wijzigen

Men kan instellen wat wel en wat niet van de geselecteerde curves moet worden weergegeven. Zo kan men de coördinaten laten weergeven, de raaklijnen aan de controlepunten op de curves en de gegevens van de controlepunten die duidelijk maken tot welke curve zij behoren en het hoeveelste controlepunt van die curve zij zijn. Aan- en uitzetten van die weergaven gebeurt door het aan- en uitvinken van de checkboxen in Choice Area –> Show. In figuur 14 zijn alle mogelijke extra weergaveopties te zien:

- ◊ de blauwe lijnstukken stellen de raaklijnen voor, als die beschikbaar zijn en/of afgeleid kunnen worden uit de controlepunten
- ◊ naast elk controlepunt geeft de bovenste regel tekst weer het hoeveelste controlepunt zij in welke curve zijn
- ◊ de onderste regel naast elke controlepunt geeft de coördinaten weer

Curve Editor - Documentatie - Inhoud



Figuur 14: Curve Editor: extra weergaveopties

IV. Beschikbare algoritmen

- ◆ Lineair (figuur 15)

Dit interpolatiealgoritme is de simpelste: per twee opeenvolgende controlepunten wordt het tussenliggend lijnstukje berekend en uitgetekend. Zo zal de curve bestaan uit de lijnstukjes van het eerste naar het tweede, van het tweede naar het derde,... en van het voorlaatste naar het laatste controlepunt.

- ◆ Bezier C0 (figuur 16)

Dit interpolatiealgoritme is de primitieve vorm van het Bezier-interpolatiealgoritme: per vier opeenvolgende controlepunten wordt een tussenliggende curve berekend als volgt: de curve start in het eerste punt in de richting van het tweede, om dan in het vierde punt te eindigen in de richting van het derde. Een curve a.h.v. het volgend viertal controlepunten wordt weer op dezelfde manier berekend, volledig onafhankelijk van het voorgaande viertal.

- ◆ Bezier G1 (figuur 17)

Dit interpolatiealgoritme is een uitgebreide vorm van het Bezier-interpolatiealgoritme: per vier opeenvolgende controlepunten wordt een tussenliggende curve berekend zoals bij de primitieve vorm. Maar de ligging van het tweede en het derde controlepunt wordt nu echter herberekend zodanig dat de berekende curve vloeidend aansluit op de berekende curve voor het vorige en het volgende viertal. Dit wordt bekomen als volgt:

Stel viertal v1 en viertal v2 opeenvolgende viertallen van controlepunten. Het laatste controlepunt van v1 en het eerste controlepunt van v2 zijn logischerwijs dezelfde: anders zouden we niet tot een aaneengesloten curve kunnen komen. M.b.v. het voorlaatste controlepunt van v1, het gemeenschappelijke controlepunt en het tweede controlepunt van v2

Curve Editor - Documentatie - Inhoud

kunnen we nu een nieuw voorlaatste controlepunt voor v1 en een nieuw tweede controlepunt voor v2 berekenen (het gemeenschappelijke blijft ongewijzigd). Dit doen we door de vector tussen het voorlaatste controlepunt van v1 en het tweede controlepunt van v2 te beschouwen; die vector verschuiven we dan zodanig dat het gemeenschappelijke controlepunt op die vector komt te liggen. De twee uiteinden van deze verschoven vector zijn dan het nieuwe voorlaatste controlepunt voor v1 en het nieuwe tweede controlepunt voor v2. Indien we nu op die manier voor elk gemeenschappelijk controlepunt twee nieuwe controlepunten berekenen, dan bekomen we een mooi vloeiende curve.

♦ Bezier C1 (figuur 18)

Dit interpolatiealgoritme is een uitgebreide vorm van het Bezier-interpolatiealgoritme: per vier opeenvolgende controlepunten wordt een tussenliggende curve berekend zoals bij de primitieve vorm. Maar de ligging van het tweede en het derde controlepunt wordt nu echter herberekend zodanig dat de berekende curve vloeiend aansluit op de berekende curve voor het vorige en het volgende viertal, én zodanig dat de berekende curve een controlepunt even snel "binnenkomt" als "verlaat", met dezelfde versnelling als het ware. Dit wordt bekomen als volgt:

Stel viertal v1 en viertal v2 opeenvolgende viertallen van controlepunten. Het laatste controlepunt van v1 en het eerste controlepunt van v2 zijn logischerwijs dezelfde: anders zouden we niet tot een aaneengesloten curve kunnen komen. M.b.v. het voorlaatste controlepunt van v1, het gemeenschappelijke controlepunt en het tweede controlepunt van v2 kunnen we nu een nieuw voorlaatste controlepunt voor v1 en een nieuw tweede controlepunt voor v2 berekenen (het gemeenschappelijke blijft ongewijzigd). Dit doen we door de vector tussen het voorlaatste controlepunt van v1 en het tweede controlepunt van v2 te beschouwen; die vector verschuiven we dan zodanig dat het gemeenschappelijke controlepunt in het midden van die vector komt te liggen. De twee uiteinden van deze verschoven vector zijn dan het nieuwe voorlaatste controlepunt voor v1 en het nieuwe tweede controlepunt voor v2. Indien we nu op die manier voor elk gemeenschappelijk controlepunt twee nieuwe controlepunten berekenen, dan bekomen we een mooi vloeiende en een in elk controlepunt even snel ingaand als uitgaand versnellende curve.

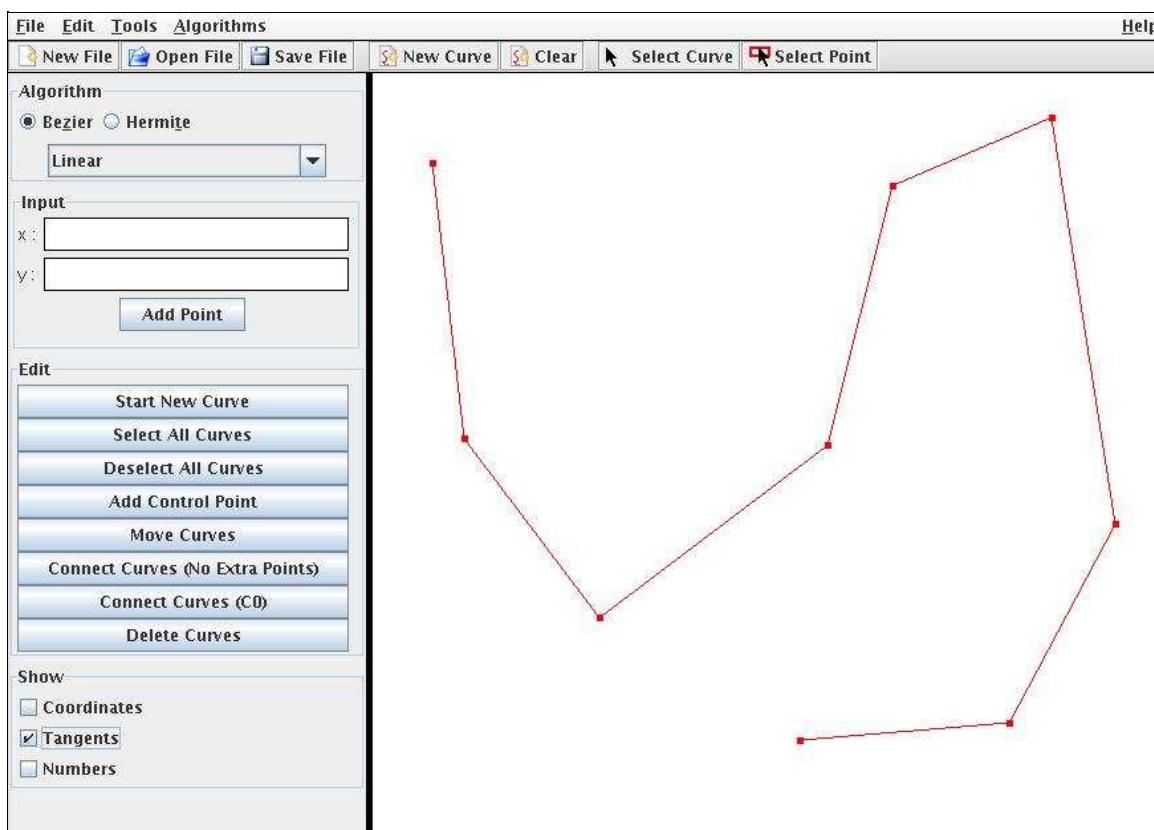
♦ Hermite normaal (figuur 19)

Dit interpolatiealgoritme is de primitieve vorm van het Hermite-interpolatiealgoritme: per vier opeenvolgende controlepunten wordt een tussenliggende curve berekend als volgt: de curve start in het eerste punt in de richting van het tweede, om dan in het derde punt te eindigen in de richting van het vierde. Een curve a.h.v. het volgend viertal controlepunten wordt weer op dezelfde manier berekend, volledig onafhankelijk van het voorgaande viertal. Het eerste controlepunt van het volgende viertal valt uiteraard samen met het derde controlepunt van het huidige viertal, om also een aaneengesloten curve te verkrijgen.

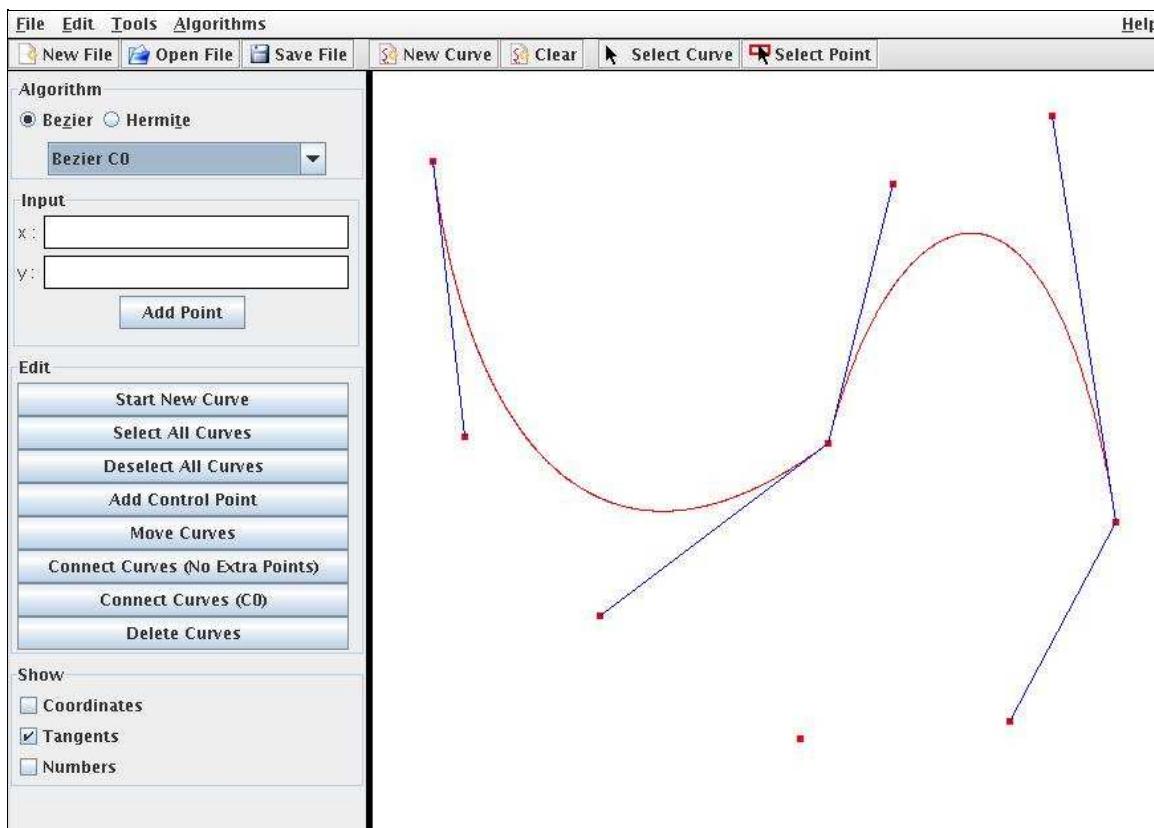
♦ Hermite Cardinal (figuur 20)

♦ Hermite Catmull-Rom (figuur 21)

Curve Editor - Documentatie - Inhoud

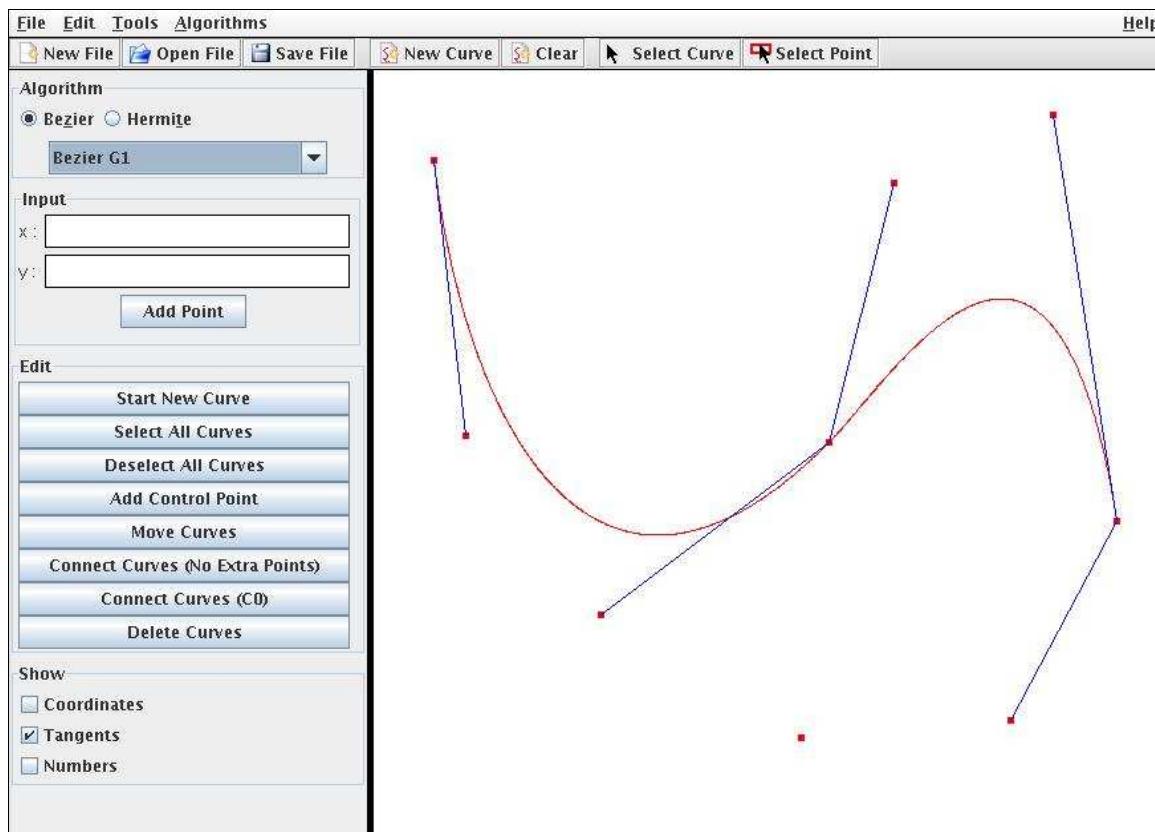


Figuur 15: Curve Editor: Lineair



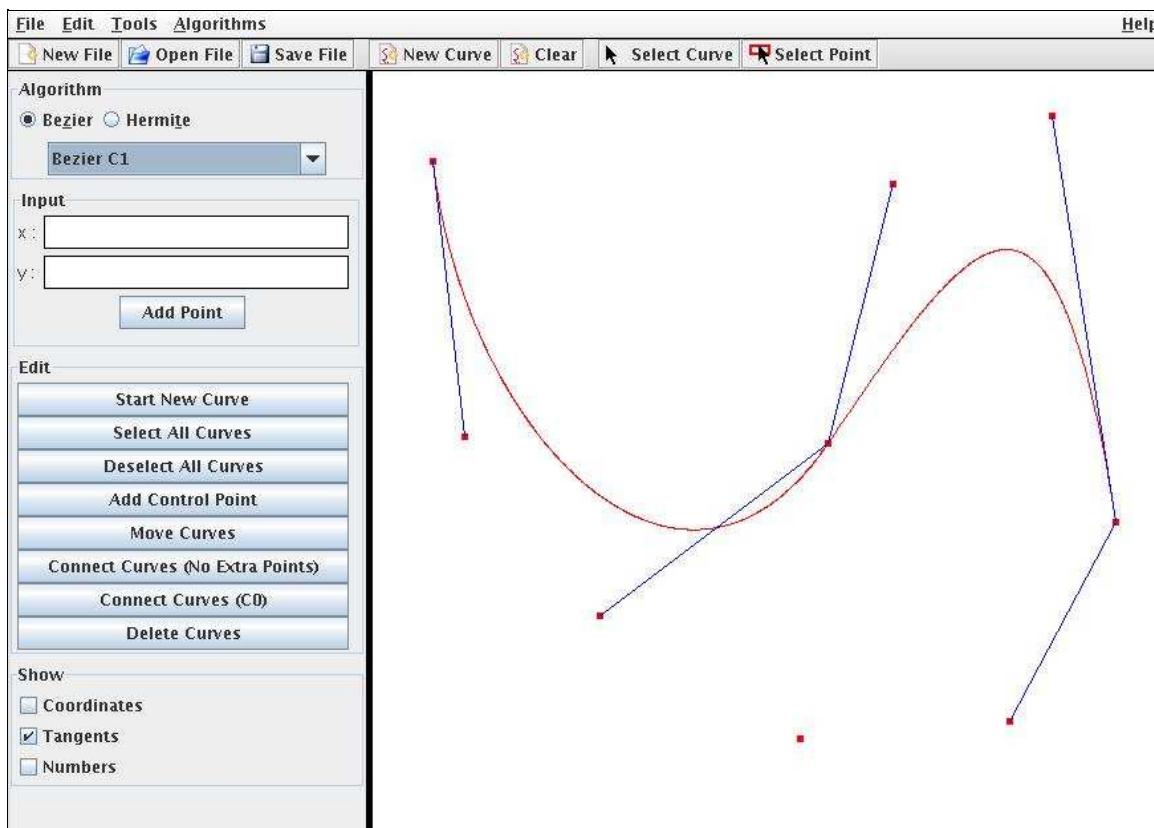
Curve Editor - Documentatie - Inhoud

Figuur 16: Curve Editor: Bezier C0

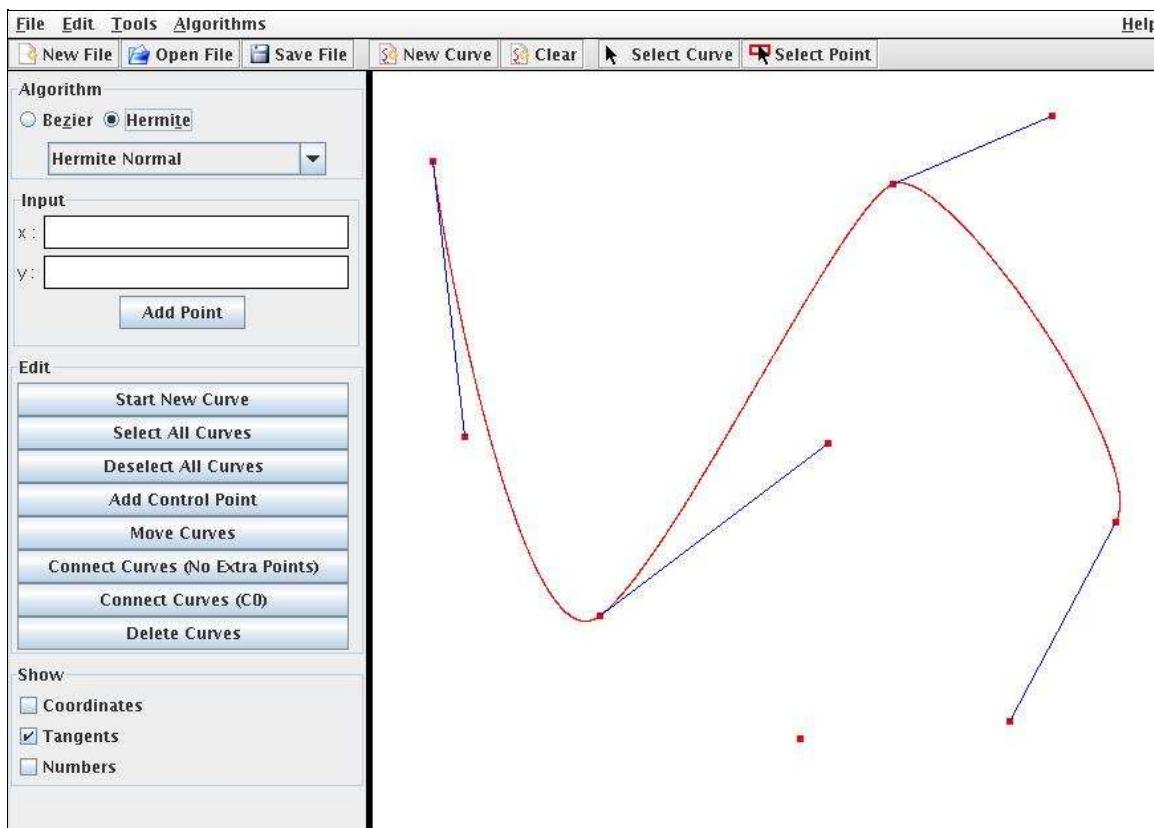


Figuur 17: Curve Editor: Bezier G1

Curve Editor - Documentatie - Inhoud

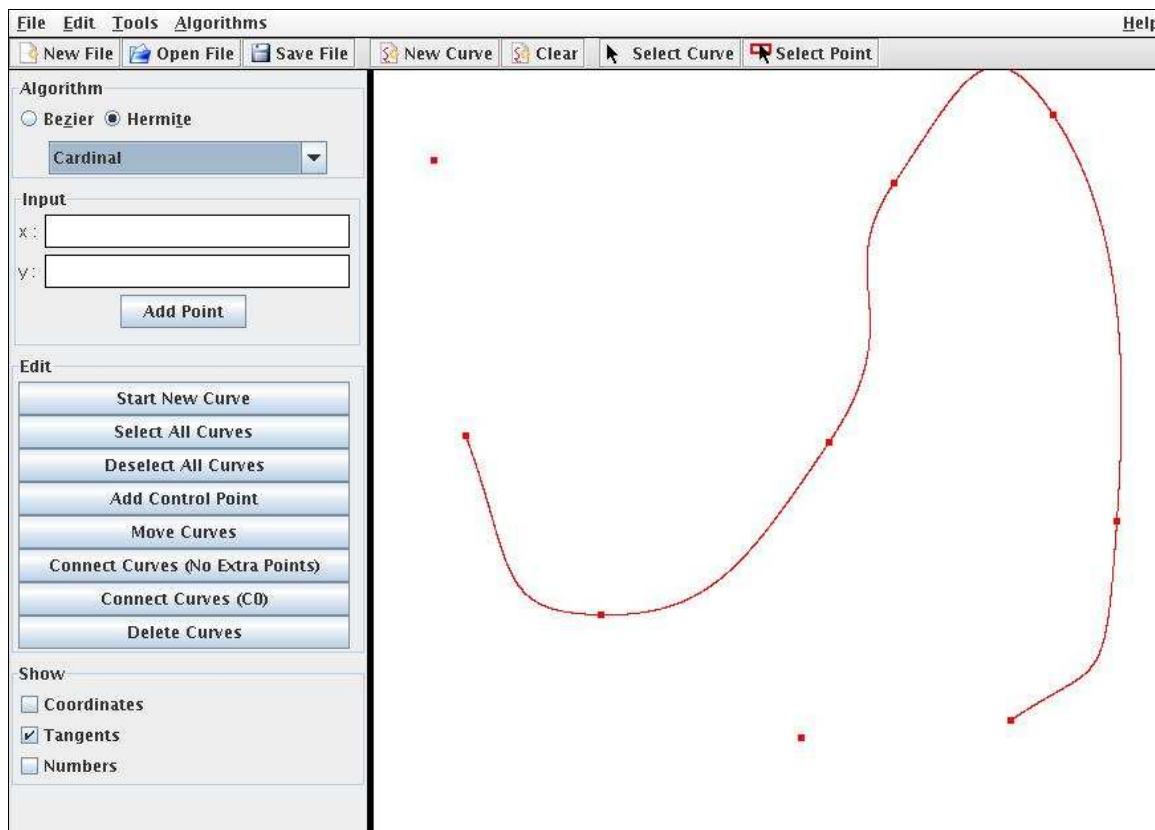


Figuur 18: Curve Editor: Bezier C1



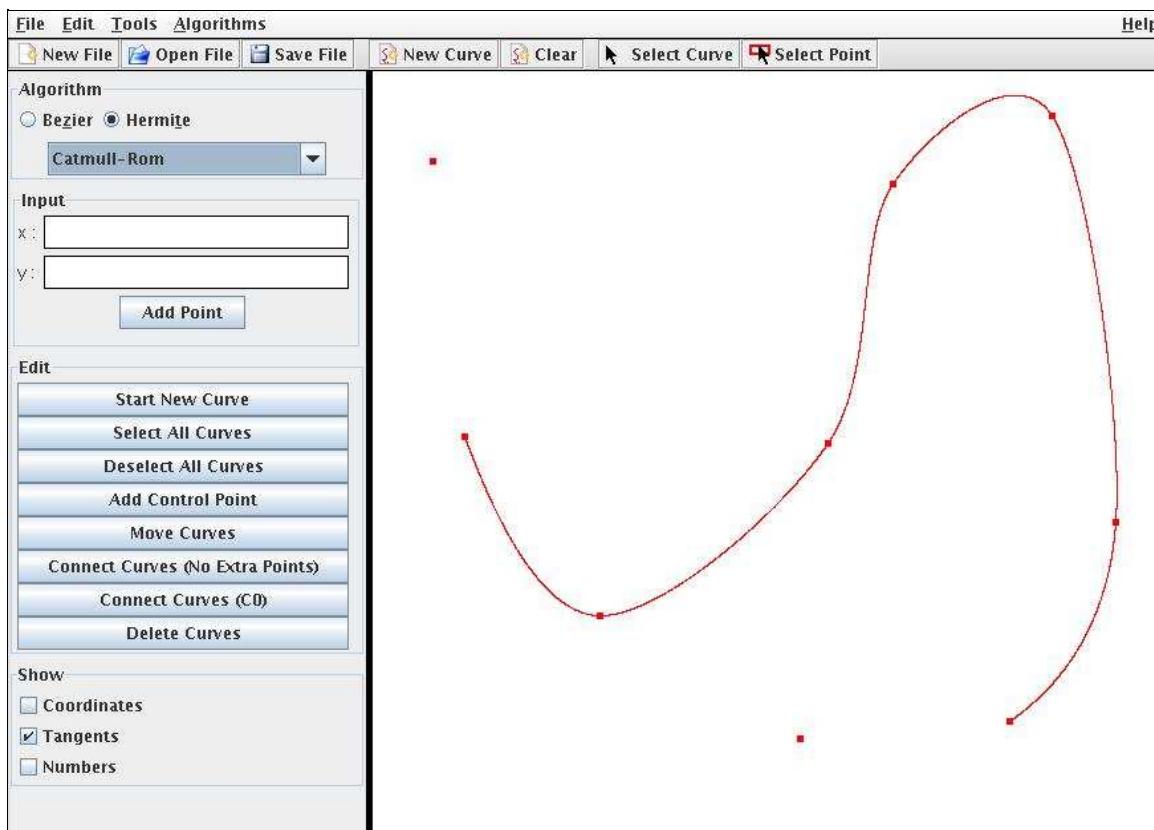
Curve Editor - Documentatie - Inhoud

Figuur 19: Curve Editor: Hermite Normaal



Figuur 20: Curve Editor: Hermite Cardinal

Curve Editor - Documentatie - Inhoud



Figuur 21: Curve Editor: Hermite Catmull-Rom

V. Ontwikkelaars

Sibrand Staessens:

Bezier-algoritmen, C1- en G1-continuïteit, selectie-functionaliteit, canvas-weergave, documentatie

Sibren Polders:

Hermite-algoritmen, file I/O, menu- en toolbar, Choice Area, Path Simulation Tool