

Project Grafische Technieken '03 - '04

Een CAGD implementatie in Java
+ Inleiding tot UML-specificatie

Heyse Caroline Mortier Maarten

1 Introductie

Dit is een verslag van het vakproject Grafische Technieken 2004, waarin de functionaliteit en architectuur van het uitgewerkte programma wordt besproken. Mede dankzij het gebruik van UML - maar ook in een poging niet te veel op de zaken uit te breiden -, werd er sterk beroep gedaan op de deugd der bondigheid in bepaalde delen; eventuele onduidelijkheden kunnen alsnog in de mondelinge presentatie worden verklaard.

Alle gebruikte *java-broncode*, *afbeeldingen* en *tekstbestanden* die bij dit project horen¹ zijn uitgewerkt door

Maarten Mortier en **Caroline Heyse**, Groep 13.

1.1 Uitvoeren

Compileren en uitvoeren gebeurt met (wanneer men zich in de map GT bevindt!)

```
javac MainGTComponent.java GrafischeTechnieken/*.java  
java MainGTComponent
```

Het programma zal automatisch met een foutmelding afsluiten als bepaalde resources (afbeeldingen of html files) niet gevonden worden.

1.2 Documentatie

Naast dit verslag is er ook online help te vinden in het programma. Deze is te vinden in het HelpVenster, een deel van de GUI. (ook te openen met menu *Help-Documentation*) Dit bevat o.a. ook een lijst van de sneltoetsen en opties van het **ConsoleVenster**, die gevoelig veranderd zijn met de oorspronkelijke opgave.

1.3 Voorzorgen

De commando's die files bewaren naar schijf zullen niet functioneren als het programma geen write-permission heeft in de huidige directory. (I/O errors gaan trouwens naar het commando-venster, als er één voorkomt bij een operatie wordt die operatie gewoon genegeerd - behalve wanneer het een nodige resource zoals een icoon of html-file betreft.) Dat zal normaal gezien wel niet gebeuren..

De directories 'curvefiles' en 'pngfiles' moeten aanwezig blijven. Hier zitten ook een paar voorbeeldcurves die kunnen worden geladen van schijf.

Wanneer men files zoals *.PNG bestanden opslaat moet men wel zelf een extensie aan de bestandsnaam toevoegen, als dat gewenst is.

¹Er werden overigens geen externe 'libraries' gebruikt.

2 Functionaliteit

Alvorens dieper in de code te duiken trachten we eerst even alle aanwezige functies, opties, en tekortkomingen op te sommen. Er is een redelijk grote kans dat enkele triviale zaken vergeten zullen worden. De geïmplementeerde extra - en speciale opties worden met hun code aangeduid.

• Functies

– Neville

Neville werd volledig geïmplementeerd; m.a.w. de gebruiker kan met een willekeurig aantal datapunten en zowel eerste - als tweede graadsafgeleiden [SPECx1] in enkele of alle punten, een Lagrange - curve construeren.

Dit algoritme werd overigens op twee fundamenteel verschillende manieren geïmplementeerd; via een recursief opwaartse of neerwaartse tactiek. Om het verschil in efficiëntie aan te tonen kan de gebruiker kiezen tussen deze twee methodes.

Het algoritme is - zoals alle recursieve algoritmes in het project (dit zal dan ook niet meer worden herhaald) - dynamisch geprogrammeerd.

– B-Spline

Men kan ook B-splines tekenen, met alle gevraagde functionaliteit; een graad wordt gekozen via één van de input-mogelijkheden in de GUI, en zodoende worden bij het aanbrengen van een aantal controle-punten een aantal splines berekend.

Verder gelden dezelfde opmerkingen als bij het algoritme van Neville.

– Bézier

Aangezien Bézier-curves niets anders zijn dan B-splines, is het natuurlijk ook mogelijk om deze curves te tekenen, door het aanbrengen van datapunten en controlepunten. De keuze van of nu een datapunt of een controlepunt wordt verwacht wordt overigens door het programma gestuurd (al kan de gebruiker dat wel pogen te 'overriden' via het ConsoleVenster.

– Input-mogelijkheden met de GUI

Men kan via de grafische user-interface (dus even die console-interface buiten beschouwing latend) :

- * Punten (data - en controle) verplaatsen, toevoegen, en verwijderen.
- * Eerste afgeleides in die punten verplaatsen en 'verwijderen'.²
- * Idem voor tweede afgeleides. [SPECx2]

²Enkel van toepassing in het Neville-algoritme natuurlijk.

- * Knopen verplaatsen, toevoegen, en verwijderen.
- * De graad wijzigen.³
- * Het soort curve wijzigen. (dit kan onder andere in het curve-select GUI element, en ook in het algemene menu)
- * De huidige curve wijzigen (zie *meerdere curves* [GENx4])
- * Inzoomen in de curve op verschillende manieren - hiervoor werd een recyclebaar systeem gebruikt zodat men kan inzoomen op het tekenvenster, op de gewichtsfuncties, en op het outputvenster. Het is mogelijk om met een zoombox te werken, een 'fit to screen' methode of een 'return to default view'.
- * Het centrum van het zichtsveld instellen (weeral is dit op een willekeurig component dat een assenstelsel gebruikt, toepasbaar).

Er zit een rudimentair help-systeem in het programma omvat (menu Help → Documentation) dat de gebruiker op weg moet zetten naar verdere bijzonderheden.

– Input-mogelijkheden met de Console

Al het voorgaande kan ook met de Console gedaan worden. Verder kan men met die Console ook nog configuraties op schijf opslaan, en een manuele **berekenCurve()** doen. (deze laatste functie wordt met de grafische interface eerder automatisch uitgevoerd en enkel op gepaste ogenblikken, met gepaste optimalisaties - als datapunten verplaatst worden bv., moeten de gewichtsfuncties niet herberekend worden).

– Outputs

Volgende zaken kan het programma effectief tonen :

- * De bekomen curve (een curve of een aantal splines). In het OutputVenster en ook in het TekenVenster. Deze curve kan bovendien geoutput worden naar een PNG afbeeldingsbestand [GUIx2], met nog allerlei parameters.
- * De BasisMatrix [GENx2], of matrices wanneer het een aantal splines betreft. In het **BasisMatrixVenster**. Wanneer er meerdere matrices zijn kan rechts de gewenste matrix worden gekozen (deze zijn voor de meeste segmenten natuurlijk hetzelfde tenzij knopen worden versleept)
- * De gewichtsfuncties [GENx1]. Opgelet, bij B-splines worden de gewichtsfuncties ook apart getekend voor elk segment, met eenzelfde 'tabbedpane' systeem. Normaal gezien tekent men deze gewichtsfuncties over het verloop van de hele knopen-vector en gebruikt men dan een 1 of 0 in de basis wanneer de waarde binnen of buiten het bereik van het huidige segment

³Van B-splines.

ligt⁴, maar wij hebben de gewichtsfuncties dus gewoon apart laten tekenen, voor elk segment. Dat komt op hetzelfde neer.. Deze curves vind men dus in het **GrafiekVenster**.

– **Meerdere curves [GENx4]**

Via een extra GUI-elementje is het mogelijk om met een aantal curves tegelijk bezig te zijn, elk van een willekeurige configuratie (neville, b-spline of bezier dus, en met andere input-data). Het soort curve wordt met kleuren codes aangeduid doorheen het hele programma (rood voor Neville, blauw voor B-spline, en groen voor Bézier), indien dit soms niet helemaal duidelijk is.

Verder in dit verslag, en op de presentatie, zal de werking van dit systeem uitvoeriger worden besproken.

– **Verder...**

Alle output-delen zijn gedouble-buffered - die dus niet moeten hertekend worden als de vensters worden verplaatst -, en er werd getracht om nooit iets teveel te berekenen (dus als maar 2 splines moeten herberekend worden zullen ze niet allemaal worden herberekend). Het is mogelijk om de *huidige* curve op schijf te slaan (er is geen optie geïmplementeerd om *alle* aanwezige curves op schijf te slaan - hoewel dat een elementaire stap is) en weer van schijf te halen (de huidige curve wordt dan vervangen door die in de file). Dit kan vanuit het menu of vanuit de console. Meer daarover in de betreffende delen.

Verdere details komen hopelijk nog naar boven.

• **Tekortkomingen/bijzonderheden**

De mogelijkheden van een GTScript werden, uit bewuste overweging, wat 'verwaarloosd'. Daardoor is een script uiteindelijk eigenlijk niets meer dan een opsomming van commandolijnen die na elkaar worden uitgevoerd. Syntax-gevoeligheid wordt ook redelijk zwaar bestraft in de meeste gevallen - een commando dat niet werd begrepen wordt eenvoudigweg genegeerd. Een commando of reeks van commando's die tot een ongeldige curve-configuratie zouden leiden, word(en) ook genegeerd en stoppen dan zelfs het hele script. Er is wel een beetje feedback omtrent het benodigd aantal parameters en andere semantiek.

Wanneer de gebruiker een datapunt wil toevoegen aan een B-spline, of dergelijke, is het script-systeem wel zo 'slim' genoeg om dit te veranderen naar een controle-punt. Dit werd gekozen omdat het toch hoege-naamd geen ambiguïteit veroorzaakt (althoewel het dan weer voor verwarring kan zorgen als de gebruiker denkt dat hij effectief een datapunt

⁴Dit is een heel ruwe uitleg

kan bepalen in een spline⁵ - daarom staat dit ook hier vermeld) Nog zo'n voorbeeld is dat, wanneer een punt wordt toegevoegd, de knopenvector zich automatisch aanpast aan de nieuwe punten-configuratie. Maar dat is ook eerder een bewuste overweging en nietigt de mogelijkheid tot het construeren van onmogelijke curves.

Verder zijn de uiterste knopen niet verplaatsbaar (dus is de 'lengte' van het kromme-bereik niet aan te passen) - maar dit leek ons eigenlijk logisch. Knopen zijn om eenzelfde logica niet over elkaar heen te verplaatsen aangezien dit tot nonsensicale resultaten zou leiden.

De klasse **Kromme** fungeert inwendig niet echt zoals verwacht, het is enkel een placeholder voor de coördinaten van de lijnsegmenten, dat als object naar de output-systemen wordt gegooid. De veeltermvergelijking die een waarde t afbeeldt op een punt⁶ zit echter vervat in de klassen **NevilleTree** en **BSplineTree**, waar er verscheidene optimalisaties en tactieken plaatsvinden die niet echt allemaal in een algemene **Kromme** leken te passen. Dat is dan echter misschien een beetje verwarrend en afwijkend van de opgave.

De multipliciteit van de relatie **GebruikersInterface** * \rightarrow **GrafischeEngine** werd omgekeerd. (dus, er is één **GebruikersInterface** die meerdere **GrafischeEngines** bevat). Dit was eigenlijk nodig om comfortabel met meerdere verschillende curve-configuraties tegelijk te kunnen werken.

De sneltoetsen voor de console verliezen hun werking in het MenuVenster. Dit is een keuze omdat de gebruiker daar alles met de muis kan doen en er eigenlijk weinig of geen nood is aan shortcuts, hij/zij heeft er ook geen controle over parameters.

De graad van een b-spline wordt niet bewaard met het 'save...' commando en wordt dus geïnitieerd op 3 bij een 'load'. Hier is geen reden voor, dit is een domme vergeetachtigheid die pas op het moment van schrijven werd ingezien.

2.1 Voorbeeldsessie

We willen de tester aansporen om van deze voorbeeldsessie (daarom niet letterlijk te volgen) gebruik te maken om een paar opties van het programma te verkennen en te ervaren hoe het met bepaalde moeilijkheden omgaat ipv. te duiken in de nogal overvloedige informatie die hierna komt.. Sorry als de toon nogal 'bevelend' klinkt het is maar een voorbeeld.

⁵Als hij/zij bijvoorbeeld denkt dat de Bezier-representatie van B-splines dit toelaat.

⁶of eigenlijk, op de gepaste lineaire combinatie van de ingevoerde punten

1. Teken een paar datapunten en trek raaklijnen in het **TekenVenster**, in curve #1 bv.
2. Gebruik het **Move** tool om een tweede afgeleide in een punt te trekken waar u een eerste afgeleide hebt ingegeven.
3. Open het **BasisMatrixVenster** en **GewichtsVenster**. Pas de grootte van alle vensters aan. Zoom ook in op het GrafiekVenster door met de muis te slepen.
4. Versleep een paar knopen door ze vast te nemen en te...ja, slepen. Analyseer alle veranderingen.
5. Open het **OutputVenster** en probeer ook hier in te zoomen (bv. een 'fit to screen')
6. Ga naar een andere curve-positie, bijvoorbeeld curve #5, die geïnitieerd staat op de B-Spline engine, en teken hier enkele punten (de graad staat geïnitieerd op 3 dus 4 punten zijn nodig om een curve te tekenen) Teken een B-spline met een paar segmenten.
7. Ga naar het menu-item 'Knopen' en verander de knopen in een open-uniforme verdeling.
8. Open het GraadManipulatieVenster via een menu (uit het hoofdmenu of uit het linker GUI-element) en verander de graad naar boven en naar beneden.
9. Verander een willekeurige curve naar een ander type (bijvoorbeeld naar een Bezier). Merk op hoe de kleurencodes in het programma meeveranderen. Teken een Bezier curve en analyseer de gewichtsfuncties.
10. Bewaar een willekeurige curve op schijf met het 'Save...'
11. Open een andere curve-positie en 'Load...' deze file terug. Merk op hoe de hele curve-configuratie, inclusief type nu in het huidige 'slot' staat.
12. Save de output naar een *.png bestand met 'Save output image...' Bekijk dit bestand in een image viewer.
13. Open het HelpVenster en gebruik deze als gids om alles wat er nu gedaan is met het **ConsoleVenster** te doen. (deze wordt geopend uit het menu 'Console')

3 Werking van de grafische engines

Dit deel bespreekt vooral hoe de boomstructuren voor de engines in elkaar steken en de motivatie tot het maken van bepaalde keuzes hieromtrent.

3.1 NevilleTree en BSplineTree

Deze klassen bevatten de piramide-structuren die voor het Neville en De Boor algoritme een representatie vormen van de bladwaarden, takfactoren, en ook de tussenwaarden (om dynamisch te kunnen programmeren is het nl. natuurlijk nodig dat niets twee keer teveel wordt berekend).

De boom kan op twee mogelijkheden worden gebruikt; opwaarts en neerwaarts.

- **Opwaartse tactiek**

1. De piramide wordt eerst geconstrueerd op zo'n efficient mogelijke manier (pointers naar kinderen worden met name gedeeld onder verschillende ouders, etc.)
2. In de blaadjes komen de X,Y waarden van de gegeven punten, afgeleiden, en tweede afgeleiden. Elke knoop bevat informatie over de blossom - of indexnotatie.⁷
3. De boom wordt dan recursief overlopen door in de takken te vermenigvuldigen met de gewenste factoren (dit verschilt iets bij Neville en De Boor natuurlijk); in de top krijgt men dan voor elke t een $P(t)$, het punt op de kromme in die knoopwaarde. Deze recursie is volledig dynamisch geprogrammeerd doordat, wanneer het algoritme langs een knoop⁸ gepasseerd is, er daar een 'flag' wordt aangezet die dat aangeeft. Wanneer het algoritme daar dan nog eens komt, weet hij dat de X,Y waarde opgeslaan in deze knoop, direct kan gebruikt worden zonder dieper af te moeten dalen.
4. Op deze manier kan men, als de densiteit bijvoorbeeld 100 lijnstuksegmenten bedraagt, met 100 wandelingen door de boom een Kromme maken.

Deze techniek is redelijk efficient, maar veel minder dan de volgende, de neerwaartse/polynomiale benadering.

- **Neerwaartse tactiek**

1. Belangrijk is hier allereerst de klasse **GTPolynomial**, een extensie op een gewone **Vector** die dus veeltermfuncties kan bevatten, vermenigvuldigen, evalueren, etc. (zie verder *UML-schema's*)
2. Deze keer wordt de boom neerwaarts overlopen. In de wortel wordt een 1 geplaatst als veelterm⁹, en dan wordt voor elk blad de resulterende veelterm berekend. Dit is opnieuw dynamisch geprogrammeerd met boolean flags dus geen enkele berekening wordt teveel

⁷Bij Neville spreekt men niet echt van blossom-notatie maar het is redelijk equivalent geïmplementeerd.

⁸'Knoop' hier dus in de betekenis van een 'boomknoop'

⁹Merk op dat dit bij B-splines enigszins anders ligt, daar wordt enkel een 1 geplaatst in de wortel van het te beschouwen segment

gedaan (er worden evenveel knopen in de boom bezocht als in de opwaartse wandeling)

3. Uiteindelijk bekomt men in elk blad een veelterm-functie van een zekere graad, bv. $t^3 + 2t^2 + 1$. Deze polynomen vormen trouwens ook de basismatrix¹⁰ en kunnen we direct gebruiken als gewichtsfuncties.
4. Met deze polynomen in de blaadjes kunnen we dan de curve construeren door voor elke t de veeltermen te evalueren en die waarde te gebruiken als coefficient van een lineaire combinatie van punten, afgeleiden, en tweede afgeleiden.

Deze methode is *veel sneller*, zeker voor hogere graden. Wanneer punten verplaatst worden hoeft ook niks opnieuw berekend te worden; de veeltermen blijven hetzelfde. Met één wandeling door de boom en 100 evaluaties van veeltermen kan dus hetzelfde resultaat als met de vorige tactiek bereikt worden, wat dus veel sneller is. Bovendien berekenen we zo simultaan de **basismatrices** en **gewichtsfuncties**! (Het algoritme van Cox - De Boor wordt overigens niet letterlijk gebruikt, alles is gebaseerd op de pyramidiale structuur.)

Het valt evenwel op te merken dat deze gewichtsfuncties ook kunnen verkregen worden als B-splines/Bézier curves op zich (zie theorie), maar dit is een omweg aangezien we ze nu toch al berekend hebben.

- **Niet geïmplementeerde optimalisaties** Er zijn nog een groot aantal verbeteringen mogelijk maar deze leken ook redelijk ingewikkeld om uit te werken. Maar we sommen toch enkele ideeën op.
 - Stel dat je een zekere boom gemaakt hebt, en je voegt een punt toe aan de curve. Dan hoeft eigenlijk niet de hele boom opnieuw geconstrueerd worden maar kun je de oude boom grotendeels behouden en enkel 'rechts' en 'boven' wat extra boominformatie invoegen. Dit lijkt echter simpeler dan het is; wanneer afgeleiden en dergelijke in het spel komen is het niet zo evident meer. Voor B-splines kon het echter met gemak worden gedaan. Het verwijderen van een punt vraagt dan weer een andere soort van update dan het toevoegen en dat wordt allemaal een beetje ingewikkeld. Vandaar hebben we dit uiteindelijk niet geïmplementeerd (B-splines zijn trouwens sowieso al efficient) *Merk wel op dat de bomen voor elk segment van een B-spline wél in elkaar haken! (identiek zoals figuur 1.61 in de cursus dus)*
 - Het is ook mogelijk om een gelijkaardig procédé toe te passen op de neerwaartse wandeling door de boom, waarbij veeltermen worden vermenigvuldigd dus, in de takken. Als die veeltermen zouden opgeslaan worden als $P(t) * Q(t)$ ipv. een enkele $P(t)$, dan zou, bij het systematisch vergroten van de boom, enkel de $Q(t)$ veranderen.

¹⁰Bij B-splines hebben we dan verschillende basismatrices

Tenminste, als de knopenvector enkel uitbreidt en niet verandert¹¹
Maar dit zou de code ook alleen maar uitbreiden. In dit geval zou het echter het algoritme van Neville gevoelig kunnen verbeteren - een algoritme dat nog altijd heel duur is, algoritmisch gezien.

- Er is ook nog iets als eindige differenties, dat in het Neville algoritme kan worden toegepast, een techniek die eigenlijk toelaat om de recursieve relatie op een andere manier te schrijven waardoor de berekeningen veel meer op elkaar voortbouwen dan traditioneel, door eerder de veranderingen te beschouwen in elke stap. Als je daarbij dan dynamisch programmeren betreft kom je waarschijnlijk dicht bij een zo optimaal mogelijk algoritme, maar deze poging werd gestaakt, grotendeels vanuit frustratie.

- **Berekening basismatrix**

De basismatrix wordt berekend door de combinatie van het bijhouden van een symbolische veeltermrepresentatie en het neerwaarts doorlopen van de boom. Zie boven voor meer uitleg.

- **Berekening gewichtsfuncties**

De gewichtsfuncties worden berekend vanuit de basismatrix. Dit kan anders (door ze door bezier-curves voor te stellen o.a.) maar dat is redelijk zinloos. Wanneer de gewichtsfuncties versleept zouden moeten worden was dit misschien wel op deze manier uitgewerkt maar dat behoorde niet tot de opties.

- **Virtuele/reele ruimte**

Gewichtsfuncties worden enkel getekend en berekend via de reele knopen, niet via de virtuele. Dit is een keuze, het is niet 'meer' werk om in de neerwaartse piramide ook de virtuele knopen te laten meespelen. Wanneer men dus bv. 'switcht' van uniforme naar open-uniforme verdelingen merk je dus niets aan de gewichtsfuncties.

3.2 Operaties

We geven een lijst van operaties, met hun parameters, die de **GrafischeEngines** kunnen uitvoeren. Elk van deze commando's is toegankelijk vanuit de console en sommige zijn exclusief daar uit te voeren. De nodige parameters (wanneer ze manueel door de user worden gebruikt) zijn ook gegeven. Hoewel dit hier niet echt thuishoort maken we van deze lijst ook gebruik om de sneltoetsen voor het console-venster al mee te geven. Alle commando's zijn *caps-insensitive*. Alle meegegeven coördinaten zijn *cartesiaanse coördinaten* (zie stuk over *AssenStelsel* en *ZoomBoxHandler*)

¹¹Dat is het geval als de knopenvector bv. van 0.01.02.03.04.05.0 naar 0.01.02.03.04.05.06.0 gaat maar niet echt wanneer een 'genormaliseerde' vector van 0.00.250.50.751.0 naar 0.00.1660.33.....

voegToePunt(double x, double y);	F1
verwijderPunt(int index);	shift + F1
verplaatsPunt(int index, double nieuwe_x, double nieuwe_y);	F2
voegToeAfgeleide(int index, double x, double y);	F3
verwijderAfgeleide(int index);	shift + F3
verplaatsAfgeleide(int index, double x, double y) = voegToeAfgeleide(...);	F3
voegToeTweedeAfgeleide(int index, double x, double y);	F4
verwijderTweedeAfgeleide(int index);	shift + F4
verplaatsTweedeAfgeleide(int index, double x, y); = voegToeTweedeAfgeleide(...); [SPECx1]	F4
voegToeKnoop(double knoopwaarde);	F5
maakUniform();	F6
maakOpenUniform();	shift + F6
wijzigGraad(int nieuwe_graad);	F7
berekenCurve();	F8
verwijderCurve();	shift + F8
veranderCurve(int nieuwe_index); [GENx4]	F9
veranderType(int nieuw_type); [GENx4]	shift + F9
save(string filename);	F11
load(string filename);	shift + F11
save2png(string filename); [GUIx4]	F12

'save' en 'load' werken zoals bewaar/laadPuntSequentie(...) etc. in de opgave, en bewaren de huidige curve, niet alle curves van de user interface. 'Load' vervangt dan de huidige curve door die die opgeslaan is in de file. Als identificatie beginnen en eindigen alle 'curve' files met de code 0xffffffffdeadbeef. Dat is natuurlijk niet echt een perfecte beveiliging tegen corrupte bestanden of bestanden die niet van het programma afkomstig zijn, er wordt gerekend op de gebruiker z'n gezond verstand wat dat betreft. (de menu-versies van deze commando's hebben trouwens een grafische file-selector)

'save2png' bewaart de output van het outputVenster naar png-formaat. De extra parameters hiervoor worden gehaald uit de huidige instellingen van het outputVenster.

Er wordt geen onderscheid gemaakt tussen 'datapunt' en 'controlepunt', de engine weet automatisch welk soort punt hij kan verwachten.

veranderType verandert het type van de huidig geselecteerde curve; de parameter hierbij is een keuze uit 0 (Lagrange), 1 (B-spline) of 2 (Bezier).

Meer 'in-depth' commentaar over het **ConsoleVenster** en de commando's aan de GrafischeEngine komt in het volgende deel.

4 ConsoleVenster

Het ConsoleVenster is in zeker opzicht naïef geprogrammeerd. Er werd geen gebruik gemaakt van Swing klassen zoals een *KeyMap* en dergelijke die het programmeren van deze dingen eleganter zou maken (uit tijdsgebrek), maar eerder met een *switch-case* en *if-else* web dat alle soorten fouten die de gebruiker kan maken met tekstinput, moet opvangen. Het valt hierbij wel op te merken dat de intelligentie van die gebruiker *heel sterk onderschat* wordt.. Er is dus redelijk veel 'parsing' code aanwezig in **ConsoleVenster** en **TextArea** (en zijn listeners) die niet zo makkelijk te begrijpen is. Er werd wel gestreefd naar een zekere low-level elegantie.

Het ConsoleVenster bestaat uit een input waar de user tekst kan invoeren, en een output die informatie geeft over de nodige syntax, over eventuele syntax-fouten, en de status van de engine. (zie *screenshots*)

Commando's zoals **voegtoePunt** trachten ook andere inputs te synchroniseren zodat dat geen verkeerde curves kunnen berekend worden. Zo zal het toevoegen van een punt ook een reel knooppunt doen toevoegen, en omgekeerd. De manier waarop dit gebeurt is wel iets verschillend maar het zou teveel tekst vereisen om dit hier uit te leggen, dat wordt geprobeerd bij de UML-schema's. Deze commando's roepen ook automatisch `berekenCurve()`, omdat dit soms op een geoptimaliseerdere manier kan gebeuren. Bv., als datapunten verplaatst worden, hoeven de gewichtsfuncties niet opnieuw worden berekend.

5 GUI

Omdat de GUI vrij complex in elkaar steekt (vooral door de hoge graad van customisatie), en de bondigheid nu al ver te zoeken is geraakt, geven we hier kort enkele (bondige) UML-schema's die de functie en architecturale positie van de meest complexe GUI elementen moeten uitleggen. Ze zijn gemaakt met **Violet**, een leuk maar esthetisch gezien lelijk programma.

*Helaas hebben we deze UML-bestanden niet in dit verslag kunnen embedden, door een fout in de compatibiliteit van de *.ps bestanden, die nu niet meer kan hersteld worden.*

Daarom printen we ze op de volgende bladen af; op de presentatie kan dan meer uitleg gegeven worden waar dit nog hoeft (over de klasse **Canvas** bijvoorbeeld waarvan het UML-schema blijkbaar gesneuveld is). Onze excuses hiervoor het is voor ons ook vervelend. Ondanks de slordigheid van dit laatste deel zit de structuur achter de GUI toch redelijk elegant in elkaar, tenminste, naar onze mening.

Aan de klassestructuur van de opgave is overigens weinig of niks veranderd (behalve de multipliciteit tussen GebruikersInterface en GrafischeEngine eigenlijk niets)

Eerst volgt een screenshot van de GUI in actie, om de vensterstructuur aan te tonen.

Daarna de UML-afbeeldingen die het gehaald hebben. In volgorde :

- Window
- GrafiekVenster + voorbeeld AssenStelselListener
- BasisMatrixVenster
- GTPolynomial