Trimesteroverschrijdend Project: Curve Editor

Groep 13: Sibrand Staessens en Sibren Polders

Donderdag 22 mei, 2008

Inhoudsopgave

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.

2 Wiskundige voorkennis

2.1 Lineaire interpolatie

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 behoefte 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:
$y_{i+1} = m.x_{i+1} + b$
$= m.(x_i + \Delta x) + b$
$= m.x_i + m.\Delta x + b$
$= y_i + m.\Delta x$
$= y_i + m$ aangezien $\Delta x = 1$ als we 1 pixel verder gaan.
Dit algoritme is duidelijk efficiënter vermits we geen 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 alzo 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:

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

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 aloritme, 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 $t_0, t_1, ..., t_m$ met $t_{i+1} = t_i + \epsilon$ voor $i = 0, 1, ..., m-1$. 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 zélf gaan berekenen als volgt:

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

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.

2.2.4 G1-continuïteit

G1-continuïteit eist dat de curves naast ononderbroken ook overal vloeiend 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. 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. Stel het 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 (zie C0-continuïteit). M.b.v. het voorlaatste controlepunt van v1, het gemeenschappelijke controlepunt en het tweede controlepunt van v2 berekent Curve Editor nu een nieuw voorlaatste controlepunt voor v1 en een

nieuw tweede controlepunt voor v2 (het gemeenschappelijke blijft ongewijzigd). Dit doet het door de vector tussen het voorlaatste controlepunt van v1 en het tweede controlepunt van v2 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 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.

2.2.5 C1-continuïteit

C1-continuïteit eist dat de curves naast ononderbroken zijn en overal vloeiend 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. 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 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 (zie C0-continuïteit). M.b.v. het voorlaatste controlepunt van v1, het gemeenschappelijke controlepunt en het tweede controlepunt van v2 berekent Curve Editor nu een nieuw voorlaatste controlepunt voor v1 en een nieuw tweede controlepunt voor v2 (het gemeenschappelijke blijft ongewijzigd). Dit doet het door de vector tussen het voorlaatste controlepunt van v1 en het tweede controlepunt van v2 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 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.

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 overzichtelijker 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 **??**. 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 **??**). 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 **??**).

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: bijhouden en picken van curves. De gebruike 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 achtten het dan ook noodzakelijk dat we een andere manier zouden vinden, die een betere balans tussen äanpassenën ïnformatie 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 excepties 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. 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, transleren, ...).

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 gevariëerd 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. 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: 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 **??**). 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 impementeren om alzo 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. 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 screenshot **??** 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 geschonken 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.

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 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.
   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 kán verschijnen.

3.3 Listeners in Java

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 gedefiniëerd 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 member-variabelen 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

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 gedefiniëerd. 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 Algorithm-functies 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, ... 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 én 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 picken 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 "wegspringen".

4 Planning

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 Handleiding

Had gij dacht ik?

B Screenshot

Gij ook? Alleja ik ga kijken welke gij al hebt en dan bijmaken wat ik nodig heb. Zodoende gaan we geen dubbel werk leveren .... "Stel niet uit tot morgen wat je vandaag door een ander kan laten doen"

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)