3D modul for syntetisk kalkulator

av

Geir Borgi
Glenn Ole Haugen
Dag Asle Johansen

Masteroppgave i
informasjons- og kommunikasjonsteknologi

Høgskolen i Agder
Fakultet for teknologi

Grimstad
mai 2006

SAMMENDRAG

ActionScript er et språk som har blitt benyttet mer og mer de senere årene og er en del av verktøyet Flash. Dette er et design verktøy som har gode metoder for animasjoner i 2D og genererer små filer som gjør det svært egnet for internett. Denne rapporten ser på mulighetene for å generere, håndtere og visualisere tredimensjonale objekter i Flash ved bruk av ActionScript. Det blir diskutert roteringsmatriser opp mot Quaternion rotasjon og isometrisk projeksjon opp mot perspektivisk projeksjon i forhold til 3D objekter. Rapporten tar for seg ulike kurver og flater og hvilke begrensninger ActionScript har ettersom kvaliteten på 3D objektene øker. Det blir også diskutert ulike metoder for fargelegging av flater med tanke på kvalitet og effektivitet samt ulike interaksjonsmuligheter som kan knyttes opp mot de projiserte objektene. Resultatene har blitt benyttet til å lage en prototyp av en 3D modul, som er implementert i en eksisterende syntetisk kalkulator utviklet i Flash.

Nøkkelord: 3D, ActionScript, Quaternion, flater

FORORD

Denne rapporten utgjør en del av utdanningskriteriene for Sivilingeniør/ Master of Science grad ved Høyskolen i Agder, fakultet for teknologi. Veileder for dette prosjektet var førstelektor Per Henrik Hogstad.

Vi ønsker å takke førstelektor Per Henrik Hogstad for god veiledning gjennom hele prosjektperioden.

Grimstad, May 2006-05-27

Geir Borgi, Glenn Ole Haugen og Dag A Johansen

INNHOLDSFORTEGNELSE

FIGURLISTE

TABELLISTE

1 INNLEDNING

1.1 BAKGRUNN

parAbel er et prosjekt som i hovedsak finansieres av Utdanningsdirektoratet og Høyskolen i Agder og gjennomføres i regi av Høyskolen i Agder.

Interessen for realfag som fordypning har gått drastisk ned de siste årene. Prosjektet har som overordnet mål å bidra til å styrke rekrutteringen til høyere utdanning i teknologi- og realfag i Norge. Sentralt i parAbel er læring av realfag via internett, og parAbel skal være et supplement til den ordinære undervisningen. Et stigende antall søkere til studier som krever realfagskompetanse er forventet som et resultat av parAbel sin satsning. Det er allerede rundt 120 skoler som benytter seg av parAbel.[1] Skottland har også slitt med noe av samme problem i at realfaginteressen og antall søkere på teknologistudier som krever realfagkompetanse har minsket. De har utviklet noe som kalles Scholar som de har hatt stor suksess med. HIA og parAbel har adoptert noen av ideene fra Scholar, og begge er bygd på samme grunnprinsipper.

Noe av hensikten til parAbel er å stimulere appetitten for både lærer og elev i realfag sammenheng. Dette skal gjøres ved å bruke moderne internett teknologi som inneholder interaktive animasjoner, eksempler og oppgaver. Oppgavene knyttes både opp mot det faglige og opp mot dagligdagse problemstillinger og teknologi.[2]

ParAbel utvikler et interaktivt simuleringsverktøy SimReal. Innenfor dette verktøyet utvikles det en syntetisk kalkulator som blant annet skal inneholde en pedagogisk interaktiv 3D modul. parAbel benytter utviklingsspråket ActionScript som plattform for den syntetiske kalkulatoren.

1.2 OPPGAVEDEFINISJON

I denne diplomoppgaven skal det utarbeides en 3D modul som genererer, håndterer og visualiserer 3D objekter til en syntetisk kalkulator ved hjelp av programmeringsspråket Actionscript. Det skal evalueres om det eksisterer begrensninger i ActionScript i forhold til 3D objekter. Videre skal det evalueres ulike pedagogiske e-læringsmetoder opp mot 3D visualisering og hvilke av disse som skal implementeres i modulen. Til slutt vil det utforskes interaktivitet mot parameteriserte 3D objekter og implementasjon av de mest hensiktsmessige løsningene.

1.3 PROBLEMSTILLING

- Hvordan skal 3D modulen skal håndtere visualisering av 3D objekter?
- Hvilke pedagogiske e-læringsmetoder lar seg implementere i 3D visualisering og hvilke av disse er mest hensiktmessig å implementere.?
- Hvordan er det mest hensiktsmessig å interaktere mot parameteriserte 3D objekter?
- Har ActionScript vesentlige begrensninger i forhold til generering, håndtering og/eller visualisering av 3D objekter?

1.4 HYPOTESER OG FORSKNINGSSPØRSMÅL

Den første hypotesen er at perspektivisk projeksjon vil være mer hensiktsmessig å benytte enn et isometrisk projeksjon. Perspektivisk projeksjon gir brukeren en dybdefølelse som er mer tilnærmet det menneskelige øyet og vil dermed gi et mer virkelig bilde av 3D objektet.

Den andre hypotesen er at Quaternion algoritmen egner seg bedre enn matriser for rotasjon av objekter i rommet ved bruk av ActionScript.

Det første forskningsspørsmålet er om for stor grad av interaktivitet vil gi mindre oversikt over parameteriserte 3D objekter. Vil for mange justerbare variabler ha innvirkning på forståelsen av problemet som 3D objektet skal vise?

Det andre forskningsspørsmålet er om programmeringsspråket ActionScript er

effektivt nok for å utvikle avanserte 3D objekter.

1.5 AVGRENSNINGER

Denne oppgaven tar for seg de pedagogiske metodene som kan fungere i en 3D modul beregnet for en syntetisk kalkulator, men ikke hvilken av de som egner seg best for læring.

Det vil ikke foreligge en evaluering av den pedagogiske effekten 3D modulen vil ha ved bruk i undervisnings sammenheng.

Pedagogisk e-læring er internettbasert læring som inneholder ulike metoder som fremmer læringseffekt og forståelse. I denne oppgaven vil det kun bli tatt hensyn til interaksjonsmuligheter og informasjonsmengde som kan fremme læringen og forståelsen.

1.6 LITTERATURSTUDIE

1.6.1 Perspektivisk - og Isometrisk projeksjon

I løpet av 90-tallet og frem til i dag har det foregått mye forskning innenfor data grafikk. Resultatet har blitt uttallige ”papers” og lærebøker. Dette gjelder både for hvordan 3D objekter skal oppfattes i rommet og hvordan 3D objekter blir generert for å virke mest troverdig i forhold til den virkelige verdenen.

Hvilken projeksjon [3] som er mest hensiktsmessig å benytte avhenger av bruksområdet. Perspektivisk projeksjon [4] er den metoden som likner det menneskelig øyet mest. Den gir en god dybdefølelse for brukeren. For å få frem denne dybdefølelsen, blir det definert ett eller flere punkter som konvergerer. Disse punktene omtales også som forsvinningspunkter. Isometrisk projeksjon [5] har ingen dybdefølelse, men har blitt brukt i stor skala i spill verdenen. *The Floating Column Algorithm for Shaded, Parallel Display of Fuction Surfaces without Patches* [6] beskriver utfyllende de ulike projeksjonene og hvordan disse kan implementeres.

Perspektivisk projeksjon

Gordons paper [7] presenterer en algoritme for optimal visualisering av overflatene til 3D objekter. Algoritmen krever lite systemressurser og er ideell for grafikkortet å håndtere. I tillegg lar den seg implementere for både perspektivisk – og isometrisk projeksjon. Tankus, Sochen, Yeshurun [8] har lagt fokuset på perspektivisk projeksjon, da dette er metoden som er tilnærmet det virkelige øyet. Her sammenlikner de perspektivisk – og isometrisk projeksjon ved rekonstruksjon av overflater på 3D objekter. Algoritmen de utviklet viste seg å visualisere 3D objektene bedre etter rekonstruksjon enn de tre eksisterende algoritmene som var laget for isometrisk projeksjon.

3D objekter er generert av utallige segmenter/ flater. Ved å finne posisjon til hvert segment/flate i rommet og normal vektoren til segmentet/ flaten, er det mulig å estimere om segmentet/ flaten syns i forhold til projeksjonen. *Estimating the Orientation of Planar Surfaces: Algorithm and Bounds* [9] og *Parametric Estimation of the Orientation of Textured Planar Surfaces* [10] viser mulige løsninger for orientering av flater i rommet ved bruk av perspektivisk projeksjon, dette gjør det mulig å ikke prosessere flater som ikke finnes i den perspektiviske projeksjon. I tillegg kan denne informasjonen bestemme hvilke farge/ skygge flaten skal ha.

Isometrisk projeksjon

Det har blitt forsket på metoder for å beregne overflaten til et 3D objekt etter transformasjon og hvordan vise denne på en mest naturlig måte på et isometrisk projeksjonsplan. I *Bending Invariant Representation for Surfaces* [11] har forfatterne utarbeidet en ny metode for håndtering av overflater etter transformasjon, basert på tidligere utviklede metoder.

Isometrisk projeksjon er en underklasse av Ortografisk projeksjon. *A Matrix-Based Approach to Reconstruction of 3D objects from Three Orthographic Views* [12] omhandler tre forskjellige Ortografiske projeksjoner og hvordan generere 3D objekter inn i disse. Paper'et utdyper en metode for å fremstille 3D objektets overflate på en effektiv måte.

1.6.2 Transformasjons algoritmer for 3D objekter

For å gi brukeren muligheter til å flytte, rotere og skalere [13], er det nødvendig å implementere algoritmer for å foreta de ønskelige transformasjonene. : *Introduction to Computer Graphics* [6], *Linear Algebra And Its Applications* [14] og [15] beskriver matrise - metoden og den grunnleggende mattematikken som ligger til grunn for de ulike transformasjonene.

A Vector Processor For 3-D Geometrical Transformations [16] beskriver en metode som minsker antall ”floating point” operasjoner som kreves for hver transformasjon og dermed øker hastigheten for hver enkelt transformasjon. På grunn av dette unngår transformasjonen unødvendig minne aksess.

En rotasjon av et 3D objekt kan medføre feil eller tap av data i 3D objektet etter transformasjon. *Efficient Geometric Transformations And 3-D Image Registration* [17] tar for seg viktigheten av å eliminere slike tap av data, dette for å unngå en kritisk feil i visualiseringen av objektet.

Quaternion

I forbindelse med rotasjon finnes det en annen metode, Quaternion [18]. *Application of Quaternions* [19] og [20] beskriver Quaternion og bruken av denne.

Kanatani [21] har studert ulike rotasjons metoder og sammenliknet disse. Hensikten var å finne den metoden som ga det mest ”korrekte” 3D objektet etter rotasjon med hensyn på tap av data.

1.6.3 Flash og Actionscript

Macromedia har blitt kjøpt opp av Adobe den 5. Desember, 2005. Adobe Macromedia selger en rekke forskjellige produkter, deriblant vektøyet Flash. Siste versjon, Flash Professional 8, med en rekke nye oppgraderinger fra forrige versjon kom ut for salg høsten, 2005. Macromedias første utgivelse kom i 1996, og var en software som originalt var navngitt FutureSplash, men ble utgitt som Flash 1.0.[22]

Det er bare i de siste utgavene av Flash, ActionScript har vært inkludert. ActionScript har et stort fokus på web-baserte løsninger og har de siste årene utviklet seg til å være med å konkurrere med språk som Java og liknende. Syntaksen i

Actionscript har også flere fellestrekk med syntaksen i Java. Ettersom ActionScript har blitt brukt mer og mer i web-applikasjoner har også videreutviklingen av språket gått raskere. Det er allikevel ikke et komplett utviklingsverktøy og på grunnlag av dette er det sannsynlig at ActionScript har begrensinger på enkelte områder.

ActionScript er skjelettet i Flash og gir muligheten til å bygge ulike objekter ved hjelp av programmering. Det har ikke alltid vært mulig å benytte objektorientert programmering i Macromedia Flash. Dette ble introdusert i versjon 2.0 av ActionScript, og var en av faktorene som økte bruksområdene til Flash.[25] Flash er et raskt språk som har lagt vekt på 2D grafikk og animasjoner. Det har på grunnlag av dette blitt omtalt som et design verktøy for web-sider. Flash har ikke lagt opp til generering av 3-dimensjonale objekter, men ActionScript gjør det mulig å generere slike typer objekter. I stedet for egne 3D objekter tilbyr Flash ulike effekter som gir en 2-dimensjonal figur et romlig bilde. Objekter med tre dimensjoner krever tyngre metoder for håndtering enn det objekter av to dimensjoner. 3D grafikk er derfor et usikkert emne i Flash. Arbeidet med versjon 3.0 av ActionScript er allerede i gang, men det er ikke kommet en dato for når denne versjonen er ferdig.

1.7 STATUS PÅ OMRÅDET

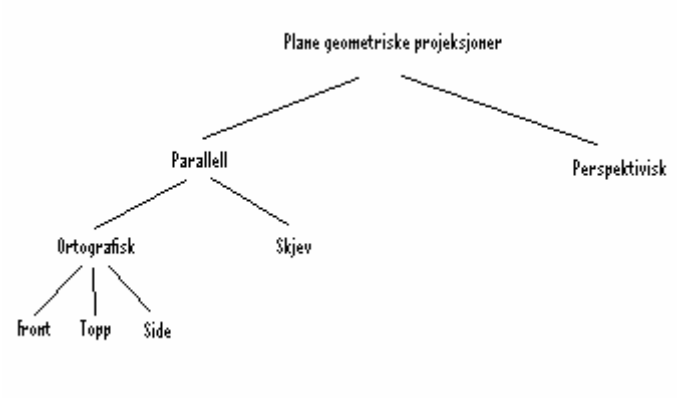
Det finnes allerede en rekke enkle 3D motorer laget ved hjelp av ActionScript, men mer avanserte 3D motorer, der det finnes flere translasjon muligheter og muligheten til å visualisere mer komplekse objekter, har det ikke blitt funnet informasjon om. Dette kan tyde på at det ikke har blitt gjort mye arbeid rundt avanserte 3D motorer i ActionScript. For de fleste av 3D motorene ligger kildekoden tilgjengelig på internett[23]. Kildekoden inneholder som oftest lite utfyllende forklaringer og ustrukturert kode, og noe koden er skrevet rett inn Flash dokumenter. Det finnes 3D motorer der rotasjonen er basert på matriser eller quaternion, men disse er enkelt oppbygd og ofte basert rundt ett objekt. Det finnes også læreprogrammer for å lage enkle 3Dmotorer, liggende ute på internett og i lærebøker, men disse tar for seg enkle objekter som består av få punkter.[24]

2 VISUALISERINGS METODER

Alle objekter som vises på en skjerm vil være 2-dimensjonale. For 2D-objekter er ikke dette noe stort problem, men for 3D-objekter må det tas hensyn til den ekstra dimensjonen. 3D-verdenen består av en x, y og z dimensjon, hvor z tilsvarer dybden. 3D-objektene og deres punkter må derfor kjøres igjennom algoritmer hvor z - verdiene beregnes inn i x og y verdiene. 3D-objektet har da blitt transformert fra en 3D-verden til en 2D-verden. Utfallet av transformasjonen avhenger av hvilken projeksjon som benyttes. Generelt sett vil en projeksjon transformere punkter i ett koordinatsystem med *n* – dimensjoner til punkter i et koordinatsystem med færre enn *n* – dimensjoner.

Projeksjonen av ett 3D – objekt blir definert av rette projeksjons stråler eller også kalt projektorer, som har utgangspunkt i ett projeksjons senter. Disse ”strålene” går igjennom hvert punkt i objektet og skjærer så ett projeksjonsplan for å forme projeksjonen. [6]

Felles for alle projeksjonene i figur 2.1 er at objektene blir projisert på ett plan istedenfor på en krummet overflate.



Figur 2.1 Undergruppene til plane geometriske projeksjoner

Perspektivisk projeksjon likner mer på hvordan det menneskelige øyet fungerer. Avstanden mellom punktene i z-retningen vil gi en dybde følelse. Punkter som er nærme vil være store, mens de som er ”lenger inn” i skjermen vil være mindre. Dette lar seg gjøre fordi projeksjonssenteret har en gitt avstand til

projeksjons planet.

Parallell projeksjon har ikke en gitt avstand mellom projeksjonssenteret og projeksjonsplanet annet enn at det er uendelig langt unna. Resultatet er at parallelle linjer vil projiseres som like lange uavhengig av hvor på z aksen de befinner seg.

Ortografiske projeksjoner har samme retning på både normalen til projeksjons planet og projeksjonen. Normalen og projeksjonen kan også ha motsatt retning.

Skjev projeksjon vil si at retningen på normalen til projeksjonsplanet og projeksjonen ikke er lik eller har motsatt retning.

Ortografisk front – elevasjon, topp – elevasjon og side – elevasjon gir ikke en 3D opplevelse av objektet. Fordelen er at avstander og vinkler kan bli målt ut i fra projeksjonen.

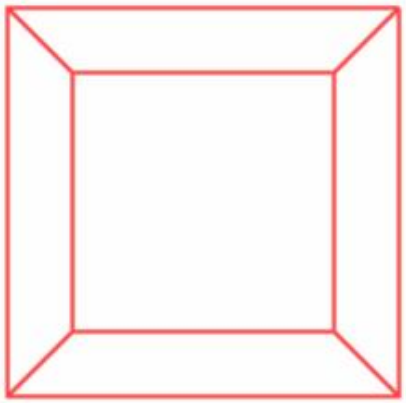
Isometrisk projeksjon bruker projeksjonsplan som ikke er normale på hovedaksene og viser dermed flere sider på en gang. Avstander kan bli målt ut i fra aksene siden parallelle linjer har lik lengde, men vinklene vil ikke kunne måles. [6]

Videre i oppgaven vil kun perspektivisk - og isometrisk projeksjon bli omtalt.

2.1 PERSPEKTIVISK PROJEKSJON

Perspektivisk projeksjon er den projeksjonen som er mest tilnærmet det menneskelige øyet. Den gir en tilsynelatende realistisk dybdeføelse. For å oppnå denne effekten er avstanden fra projeksjonssenteret definert, i tillegg til at ett eller flere forsvinnings punkter er gitt. Et forsvinningspunkt bidrar til at et sett av parallelle linjer som ikke er parallelle med projeksjons planet konvergerer mot dette punktet. Antall forsvinnings punkt bestemmes ut fra hvor mange akser projeksjons planet skjærer.[6]

Projeksjonen som vises i figur 2.2 benytter en ett – punkts perspektivisk projeksjon. Dybde følelsen kommer her tydelig frem.

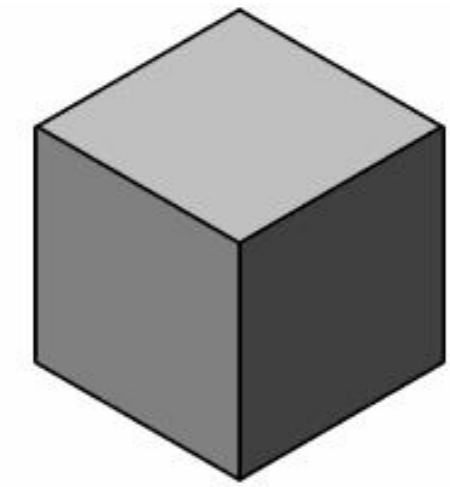


Figur 2.2 Ett-punkts perspektivisk projeksjon av en kube

2.2 ISOMETRISK PROJEKSJON

Isometrisk projeksjon bevarer lengden til linjer som er parallelle med projeksjonsplanet. En viktig egenskap til denne projeksjonen er at alle hovedaksene er like mye forkortet. Det vil si at målinger langs aksene benytter samme målestokk Linjer som ikke er parallelle med projeksjonsplanet vil da være enkle å beregne lengden til. Isometrisk projeksjon viser ikke korrekte vinkler i forhold til det originale objektet.

Figur 2.3 viser en kube som er projisert ved hjelp av isometrisk projeksjon. Her er det lett å se at det er en kube med rette vinkler, da alle parallelle linjer projiseres med lik forkortning.



Figur 2.3 Isometrisk projeksjon av en kube

2.3 PEDAGOGIKK OG VISUALISERING

Matematiske funksjoner som danner objekter i rommet er vanskelige å kunne se for seg når en bruker penn og papir. Her vil både den perspektiviske – og isometriske projeksjonen gi brukeren et bilde av hvor objektet er i rommet og hvordan det ser ut. Videre kan brukeren knytte for eksempel beregninger av topp-/bunn - punkter opp mot projeksjonen, som kan bidra til en bedre forståelse av problemet.

En kan gå enda ett skritt videre hvor brukeren interaktivt jobber opp mot projeksjonen for å gjøre ulike beregninger på den matematiske funksjonen. I dette tilfelle er det ikke sikkert begge projeksjonene som er nevnt vil egne seg like bra. Antakeligvis vil den isometriske projeksjonen egne seg bedre siden det er lik målestokk langs alle aksene og at parallelle linjer har lik forkortning.

2.4 ACTIONSCRIPT OG VISUALISERING

ActionScript har ingen innebygde metoder for å vise ulike projeksjoner av ett og samme objekt. Dette vil være helt opp til utvikleren selv å implementere.

ActionScript 2.0 har innebygd en klasse som heter movieClip. Denne klassen definerer hvordan punkter, linjer, objekter osv. tegnes opp og hvordan de vises for brukeren. Koordinatsystemet til hvert movieClip har origo oppe til venstre, positiv x-akse går mot høyre og positiv y akse går nedover. I denne applikasjon finnes det to koordinatsystem, ett for movieClipet og ett for objektet. Koordinatsystemet for objektet har samme origo som movieClipet når objektet blir generert. Z aksen har positiv retning inn i skjermen, x aksen har positiv retning til høyre og y aksen har positiv retning oppover.

Projeksjonssenteret ligger på normalen til movieClipet ut av skjermen og treffer origo til koordinatsystemet til objektet. Projeksjonssenteret flyttes eller roteres ikke etter at objektet er tegnet opp, men objektet med sitt koordinatsystem vil være mulig å rotere etter brukerens ønske.

Ved bruk av perspektivisk projeksjon kan brukeren selv definere avstanden fra projeksjonssenteret til projeksjonsplanet. Projeksjonsplanet skjærer z aksen i z = 0 og er parallell med x og y aksen. For å gjennomføre projiseringen, beregnes de nye x og y verdiene ut i fra de opprinnelige x og y verdiene samt z verdien til hvert punkt og avstanden fra projeksjonssenteret til projeksjonsplanet. I formlene under vil d

representere avstanden fra projeksjonssenteret til projeksjonsplanet. Formlene under gir den perspektiviske projeksjonen av x verdien til punktet og y verdien.

$$x_p = d * \tan(\tan^{-1}(\frac{x}{d-z}))$$

Formel for beregning av den projiserte x verdien til et punkt.

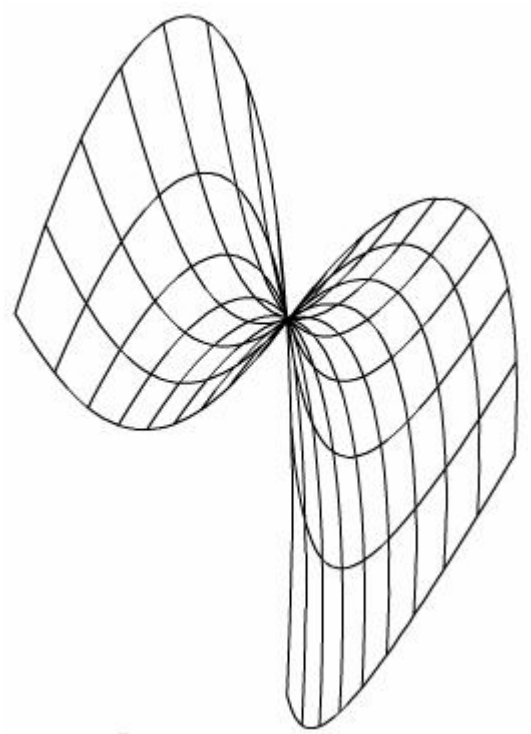
$$y_p = d * \tan(\tan^{-1}(\frac{y}{d-z}))$$

Formel for beregning av den projiserte x verdien til et punkt.

Den isometriske projeksjonens projeksjonssenter er uendelig langt unna. Projeksjonsplanet er likt som for perspektivisk projeksjon. Siden y-verdien til hvert punkt blir kalkulert ved hjelp av x og z verdien, kan vi benytte x og y verdien til hvert punkt direkte i projiseringen. Ingen ny beregning er da nødvendig.

2.5 PERSPEKTIVISK MOT ISOMETRISK

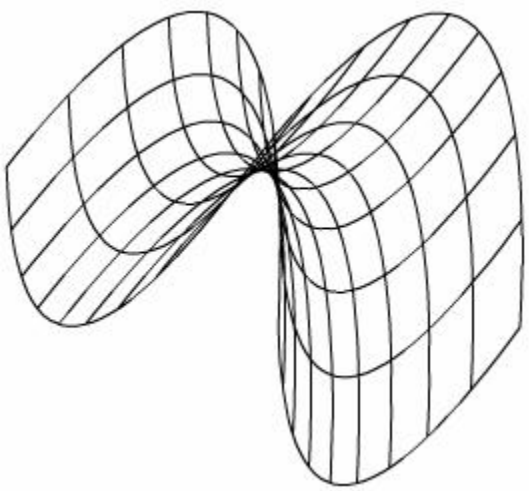
Perspektivisk projeksjon er den som likner mest på det menneskelig øyet. For å få dette til er det definert et projeksjonssenter en gitt avstand fra projeksjonsplanet. I tillegg blir det definert ett antall forsvinningspunkt bestemt av hvor mange akser projeksjonsplanet skjærer. Parallelle linjer som ikke er parallelle med projeksjonsplanet konvergerer mot disse forsvinningspunktene. I figur 2.4 vises en perspektivisk projeksjon av et objekt.



Figur 2.4 Perspektivisk projeksjon

Parallelle linjer som har ulik avstand til projeksjonsplanet vil ha forskjellig forkortning. I en matematisk sammenheng kan dette medføre feiltolkninger av 3D objektet. Samt det vil være vanskelig å gjøre målinger ut i fra aksene som tilhører koordinatsystemet som 3D objektet er tegnet opp i. Et eksempel på dette er, hvor z aksen har positiv retning inn i skjermen, projeksjonsplanet skjærer kun z aksen og forsvinningspunktet ligger på z aksen. Da vil punktet p(10, 0, 20) være nærmere y aksen, slik brukeren vil se det, enn punktet r(10, 0, 10). Dette er fordi punktet r ligger nærmere projeksjonssenteret enn punktet p. I dette eksempelet vil det være vanskelig for brukeren å se at linjen mellom punkt p og r er vinkelrett på x aksen.

Isometrisk projeksjon har et projeksjonssenter som er uendelig langt unna, som vil si at linjer vinkelrett på projeksjonsplanet er parallelle. I tillegg finnes det ikke forsvinningspunkt(er). Det betyr at parallelle linjer, som ikke er parallelle med projeksjonsplanet, blir forkortet like mye. I eksempelet over vil det være enkelt å se ut i fra aksene at punktene p og r danner en vinkelrett linje på x aksen. Figur 2.5, vist på neste side, illustrerer samme objekt som i figur 2.4, men her er det benyttet isometrisk projeksjon.



Figur 2.5 Isometrisk projeksjon

Ett annet aspekt som fremhever isometrisk fremfor perspektivisk projeksjon er kalkulering av punktene som danner projeksjonen. En detaljrik flate som skal projiseres krever mye fra CPUen og da er det viktig å ha så få kalkuleringer som mulig for å effektivisere for eksempel rotering. Siden y verdiene til hvert punkt blir kalkulert ved hjelp av x og z verdiene, kan x og y verdiene benyttes direkte inn i en isometrisk projeksjon. For en perspektivisk projeksjon må de projiserte x og y verdiene bli beregnet ut fra de opprinnelige x og y verdiene. Det trengs færre kalkuleringer for å vise en isometrisk projeksjon enn en perspektivisk projeksjon.

3 KURVER

Bezier kurver ble kjent 1962 da den franske ingeniøren Pierre Bezier benyttet denne type kurver til å designe bil karosseri. Kurvene ble utviklet i 1959 av Paul de Casteljau.[26] Bezier kurvene blir brukt til å danne grunnlag for formen til en kurve.

3.1 LINEARE BEZIER KURVER

Lineare kurver betegnes som en rett linje fra ett punkt til et annet. En lineær Bezier kan beskrives som:

$$B(t) = (1- t)^2P_0 + t*P_1 , t \in [0,1]$$

der P_0 og P_1 start og endepunktene til linjen.

3.2 QUADRATIC BEZIER KURVER

En quadratic Bezier kurve består av tre punkter. To av disse punktene blir kalt ankerpunkter og definerer start og endepunktet til kurven. Det tredje punktet kalles et kontrollpunkt og definerer hvordan kurvingen skal være. På en Bezier kurve vil alltid ankerpunktene bli interpolert. Dette vil si at kurven treffer og går igjennom disse punktene. Kontrollpunket vil derimot ikke bli interpolert, men kun bestemme vektingen av kurven. Dette vil si at kurven ikke vil ha en skarp overgang eller brytning i kontroll punktet, men at kurvingen vil ha en dragning mot punktet. Siden kurven interpolerer start og endepunktet vil det være enkelt å sette sammen flere kurvesegmenter til en lengre og mer kompleks kurve. Når en kurve starter i samme punkt som en annen kurve sluttet i, vil det si at dette er et knutepunkt med C0 kontinuitet. En quadratic Bezier kurve kan uttrykkes som:

$$B(t) = (1 – t)^2P_0 + 2t(1 – t)P_1 + t^2P_2 , t \in [0,1]$$

der P_0 og P_2 er ankerpunkter og P_1 er kontrollpunktet.

3.3 CUBIC BEZIER KURVER

Cubic Bezier kurver er en utvidelse av quadratic Bezier kurver. Denne typen kurve består også av to ankerpunkter, men har to kontrollpunkter i stedet for ett. Dette gir kurven en større fleksibilitet og gjør det enklere å forme kurven på grunn av at det eksisterer to kontrollpunkter som vekter retningen på kurven. To kontrollpunkter på samme sted vil gi en spissere bue, som i motsetning til to diagonalt motsatte kontrollpunkter vil gi en s-formet kurve. På samme måte som quadratic Bezier er det enkelt å oppnå en C0 kontinuitet, men gir også mulighet for å enklere oppnå C1 kontinuitet uten å legge for store begrensinger på kurvens fleksibilitet. C1 kontinuitet oppnås ved at et knutepunkt har lik tangent til de to nærmeste kontrollpunktene. Dette vil si at knutepunktet må ha lik tangent fra det siste kontrollpunktet til endepunktet på den første kurven som fra startpunktet til det første kontrollpunktet på den andre kurven. Denne typen kurve kan uttrykkes som:

$$B(t) = P_0(1-t)^3 + 3P_1t(1-t)^2 + 3P_2t^2(1-t) + P_3t^3 \text{ , } t \in [0,1]$$

der P_0, P_3 er ankerpunkter og P_1, P_2 er kontrollpunkter.

3.4 ACTIONSCRIPT OG KURVER

Før en begynner å generere større 3-dimensjonale objekter, er det nødvendig å se på hvilke metoder ActionScript tilbyr for å tegne opp segmenter på skjermen. Actionscript har definert to muligheter for å generere linjer og kurver. Disse to metodene har kalles ”*lineTo()*” og ”*curveTo()*”.

lineTo() genererer en rett linje og tar et argument som definerer hvor endepunktet til denne linjen er.

curveTo() genererer en kurve. Den tar to argumenter inn, som definerer sluttpunktet til kurven og et kontrollpunkt. Dette kontrollpunktet styrer retningen på buen til kurven og blir ikke interpolert. Dette vil med andre ord si at curveTo() er en kurve av typen quadratic Bezier.

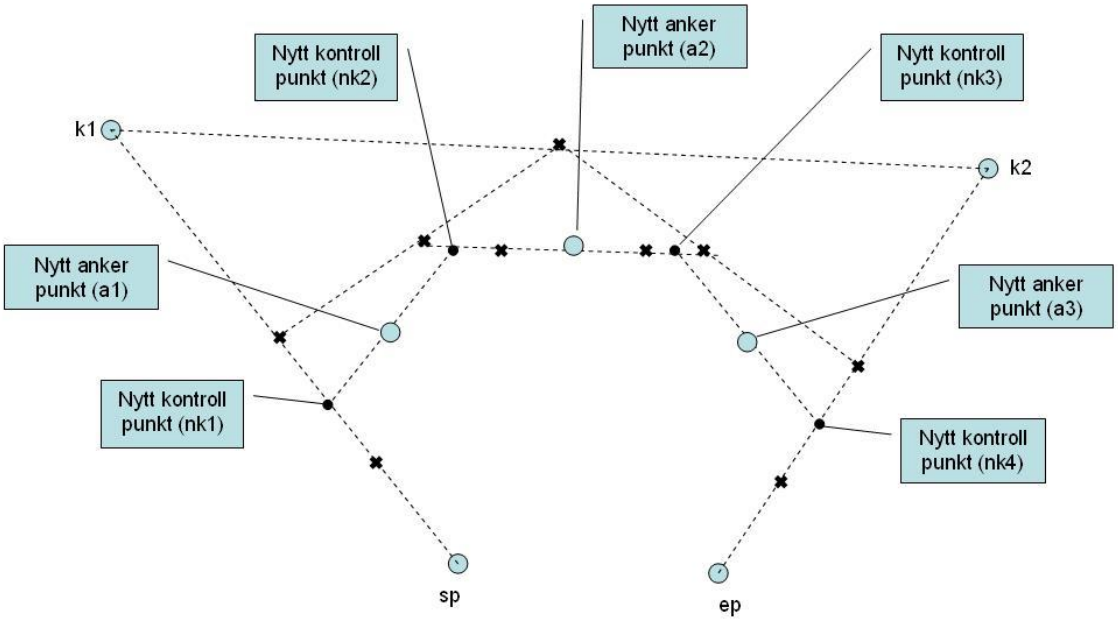
Quadratic Bezier kurver har begrenset utformingsmuligheter siden det kun eksisterer ett kontrollpunkt. Det vil selvfølgelig være mulig å generere en avansert kurve ved hjelp av mange små quadratic kurver, men dette ville være lite effektivt.

Dette kommer klarere frem dersom det skal genereres flere mindre flater som sammen danner en større flate. Det vil derfor være ønskelig å kunne benytte cubic Bezier kurver i ActionScript.

Det finnes flere metoder for å generere en cubic Bezier kurve. Det er mulig å benytte ligningen for en cubic Bezier, men siden dette krever en forholdsvis tung utregning vil det være hensiktsmessig å se på andre muligheter. Det finnes flere metoder å generere en cubic Bezier, men de fleste av disse kurvene vil være enten være nøyaktige og tunge å kalkulere, eller mer unøyaktige men krever mindre kalkuleringer. Eksempler på disse metodene kan være tangent metoden, B-splines metoden og midtpunktsalgoritmen. I en artikkel av Timothée Groleau[27] ble det gjort vurderinger på hvilke av disse metodene som egner seg best for implementasjon i en pc-applikasjon med tanke på effektivitet og nøyaktighet. I denne artikkelen kom det blant annet frem at midtpunktsalgoritmen ved bruk av fire quadratic Bezier kurver var en forholdsvis nøyaktig og effektiv metode for generering av cubic Bezier kurver.

3.4.1 Midtpunktsalgoritmen

Denne metoden går ut i fra fire punkter som definerer en cubic Bezier kurve. På figur 3.1 er start og endepunktene definert som sp og ep. De to kontrollpunktene er definert som k1 og k2. Denne metoden bruker kun enkle beregninger og blir derfor effektiv for implementering, men er noe unøyaktig.



Figur 3.1 Bruk av midtpunktsalgoritmen for kalkulering av kubisk Bezier vha fire quadratic Bezier kurver

Det har nå blitt generert nok punkter til å definere følgende quadratic Bezier kurver:

- Ankerpunkter sp og a1 med nk1 som kontrollpunkt.
- Ankerpunkter a1 og a2 med nk2 som kontrollpunkt.
- Ankerpunkter a2 og a3 med nk3 som kontrollpunkt.
- Ankerpunkter a3 og ep med nk4 som kontrollpunkt.

Disse fire quadratic Bezier kurven vil nå danne en cubic Bezier kurve.

3.5 PEDAGOGIKK OG KURVER

Bezier kurver er enkle å manipulere siden de både har interpolerte ankerpunkter og kontrollpunkter som ikke er interpolert. Ved å gi brukeren mulighet til endre disse punktene og dermed endre formen på kurven, vil dette kunne gi en større forståelse av hvordan vektingen til kontrollpunktene fungerer. For å gå over på matematiske funksjoner vil det være interessant å følge kurven ved hjelp av

koordinater ettersom brukeren beveger seg langs kurven. Det kan være enklere for enkelte å prosessere tall i stedet for en tegnet kurve. Ved bruk av både visualisering av kurven og mulighet for å få tak i koordinatene til hvert punkt på kurven, vil brukeren kunne kombinere disse. Videre vil det være mulig å visualisere og stille på tangenten til et punkt på kurven. Her ligger det en liten utfordring, for ved bruk av lineTo() metoden i ActionScript tegnes det små rette linjestykker som sammen danner en linje. Problemet er at disse linjestykkene er svært små på en detaljrik kurve. Det kan derfor virke forvirrende dersom flere punkter ser ut til å ha samme tangent. Det bør derfor eksistere egne intervaller for hvilke punkt som skal vise tangent og dette intervallet må eksistere i forhold til hvor detaljert kurven er fremstilt.

3.6 DISKUSJON

3.6.1 lineTo() opp mot curveTo()

lineTo() er en enkel metode, men er svært fleksibel og kan benyttes til å visualisere alle typer objekter. lineTo() tegner kun et rett linjestykke og dette vil si at det må benyttes flere lineTo() operasjoner for å danne en kurve. Representasjonen av kurven blir jevnere jo flere linjesegmenter som blir benyttet. Representasjon av en jevn kurve har både en positiv og en negativ effekt. Det positive er at det er mulig å representere en kurve svært nøyaktig og dermed en korrekt representasjon. Det negative er derimot at det kreves mer tid å kalkulere og tegne opp mange linjestykker enn å kalkulere og tegne opp få linjesegmenter med større lengde. curveTo() metoden har samme egenskaper som en quadratic Bezier og den fungerer derfor bra til denne typen kurver som benytter et kontrollpunkt som ikke blir interpolert. Samtidig innebærer dette restriksjoner når andre typer kurver skal representeres. Egenskapen med at kontrollpunktet ikke blir interpolert kan skape noen problemer. Dersom en kurve skal representeres ved hjelp av curveTo() metoden, og kurven ikke er av typen Bezier, vil det være vanskelig å anslå hvilket punkt som skal benyttes til kontrollpunkt siden dette er et punkt som ikke ligger på kurven. For at kurven skal bli representert korrekt må kurven som genereres av curveTo() være så liten at kontrollpunktet kan legges rett utenfor et punkt som kurven skal gå igjennom. Dette vil igjen si at kurveenhetene begynner å bli så små at curveTo() kurven nærmer seg en rett linje.

Så hva er da mest hensiktsmessig å bruke av curveTo() og lineTo()? Dersom det dreier seg om en jevn quadratic Bezier kurve vil det være mest hensiktsmessig å benytte curveTo(), siden dette kun krever en operasjon. Dersom en kurve skal genereres ut i fra et matematisk uttrykk er det nødvendig å klargjøre graden av korrekt representasjon som kreves. For å beregne hvilke punkter som kurven representeres, gjøres dette ved å la x variere med hensyn på y eller omvendt. For å kunne benytte curveTo() må applikasjonen tolke de to neste punktene på kurven før metoden kan brukes. Deretter må det tolkes hvilken retning kurvingen av kurven har, for å bestemme hvor kontrollpunktet skal ligge. Dette krever mer beregning enn det som kreves av lineTo(), der lineTo() metoden kan kjøres ettersom hvert punkt for kurven blir regnet ut. Hvis det derimot ikke kreves en nøyaktig opptegning av kurven er det mulig å hoppe over flere kalkuleringspunkt, det vil si å for eksempel regne ut y verdien til kurven for hver 30 x verdi. lineTo() vil i dette tilfelle tegne en rett linje mellom punktene og gi et uakseptabelt resultat, dersom linjesegmentet som genereres skal ha kurving. curveTo() har mulighet til å gi kurvesegmentet en kurving. Selv om denne kurvingen ikke er helt korrekt, vil formen til segmentet ikke ha like stort avvik som ved bruk av lineTo(). I dette tilfellet vil derfor curveTo() være å foretrekke. Denne oppgaven går ut på å lage en 3D modul for en kalkulator. Oppgaven til modulen skal da være å visualisere et matematisk uttrykk og da vil det være essensielt at dette uttrykket blir presentert korrekt. Den beste løsningen vil derfor være å benytte lineTo() metoden siden denne metoden både krever færrest kalkulasjoner og gir en mer korrekt representasjon.

3.6.2 Quadratic Bezier opp mot cubic Bezier

Hovedoppgaven til Bezier kurvene i 3D modulen er å generere egendefinerte flater som er av matematisk interesse. Både quadratic Bezier og cubic Bezier kan brukes til å generere tredimensjonale flater, men hvilken av disse egner seg best? En cubic Bezier har den fordelen at den benytter to kontrollpunkter, som gjør formen til kurven mer fleksibel i forhold til en quadratic Bezier. Dette fører til at det ikke er nødvendig med like mange flateenheter for å generere den ønskede flaten, i motsetning til bruken av quadratic Bezier flater. Det vil derfor være enklere å bruke cubic Bezier til generering av egendefinerte flater.

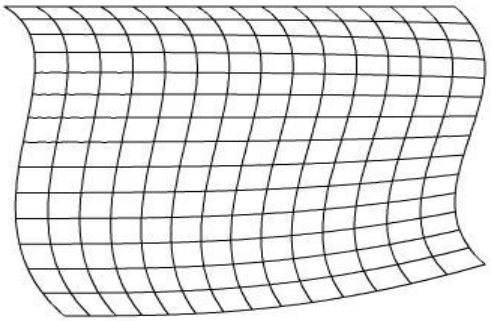
3.6.3 Midtpunktsalgoritmen opp mot ligning for cubic Bezier

Selv om midtpunktsalgoritmen er en enkel og effektiv metode for generering av cubic Bezier har den også noen ulemper. For det første er den noe unøyaktig. Det er mulig å beregne flere midtpunkt og benytte flere quadratic Bezier kurver for å simulere en cubic Bezier kurve optimalt, men da ville effektiviteten forsvinne og en kunne like godt representert kurven ved hjelp av den matematiske formelen for cubic Bezier kurver. En annen ulempe er bruken av curveTo() metoden i ActionScript. Den er enkel og effektiv, men det er vanskelig å få tak i hvilke punkter kurven går i gjennom. Dette vil si at hvis det er ønskelig å følge punktene til kurven, må disse punktene beregnes ut i fra formelen for quadratic Bezier. Samtidig må en holde orden på hvilket kurvesegment som er gjeldende. Denne metoden for å representere en cubic Bezier kurve vil derfor egne seg best som ren grafisk fremstilling, samt noen matematiske operasjoner som endring av kontroll og ankerpunkt.

4 FLATER

4.1 CUBIC BEZIER FLATER

Det har tidligere i denne rapporten blitt diskutert at det ville være mest enklest å generere flater ved hjelp av kubiske Bezier kurver i stedet for quadratic Bezier kurver. Det er derfor ønskelig å teste ut hvordan Flash takler en enkel cubic Bezier flate. En cubic Bezier flate blir bygd opp av flere cubic Bezier kurver. Se figur 4.1.



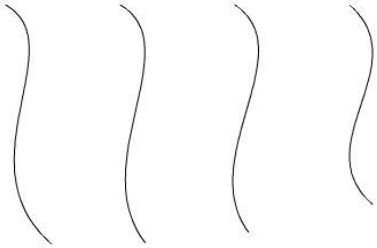
Figur 4.1 som er generert med hensyn på bærekurver

Under testing ble det benyttet fire cubic Bezier kurver til å danne grunnlag for en flate. Den parametriserte formelen som ble benyttet var:

$$X(s,t) = (1-s)^3 (P_{11x}(1-t)^3 + 3P_{12x}(1-t)^2t + 3P_{13x}(1-t)t^2 + P_{14x}t^3$$
$$+ 3(1-s)^2s(P_{21x}(1-t)^3 + 3P_{22x}(1-t)^2t + 3P_{23x}(1-t)t^2 + P_{24x}t^3$$
$$+ 3(1-s)s^2(P_{31x}(1-t)^3 + 3P_{32x}(1-t)^2t + 3P_{33x}(1-t)t^2 + P_{34x}t^3$$
$$+ s^3(P_{41x}(1-t)^3 + 3P_{42x}(1-t)^2t + P_{43x}(1-t)t^2 + P_{44x}t^3)$$

[6]

der P_{11} til P_{14} representerer en cubic Bezier kurve. Denne formelen ble brukt for X(s,t), Y(s,t) og Z(s,t). Det ble derfor generert fire kurver som skal danne bærekurvene for flaten. Disse er avbildet på figur 4.2.

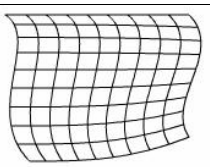


Figur 4.2 Bærekurvene til den kubiske Bezier flate

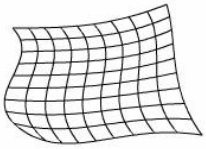
En enkel Bezier flate har allikevel ingen stor nytte. Det som vil være interessant er å se på er hvordan Flash reagerer når det blir satt sammen flere Bezier flater for å generere en større flate.

Det ble derfor generert fire flater som har kontakt med hverandre. Dette vil si at to flater med kontakt er definert ut i fra noen felles punktsett. Punktsettet til disse fire flatene, er definert ut i fra fire bezier kurver for hver flate. Hele flaten vil derfor være definert av 16 cubic Bezier kurver. Bezier kurvene som er benyttet er vist i tabellene 4.1, 4.2, 4.3 og 4.4. Figur 4.3 Viser flatene satt sammen.

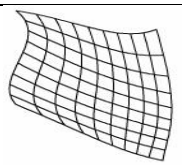
Tabell 4.1 Første Bezier Flate

Bezier type	Ankerpunkt1 (x, y, z)	Kontrollpunkt1 (x, y, z)	Kontrollpunkt2 (x, y, z)	Ankerpunkt2 (x, y, z)	Bilde
cubic	50, 100, 200	80, 100, 200	30, 180, 200	70, 220, 200	
cubic	100, 100, 170	130, 120, 170	90, 180, 170	110, 220, 170	
cubic	150, 100, 150	180, 120, 150	130, 180, 150	150, 220, 150	
cubic	200, 100, 100	230, 130, 100	180, 170, 100	200, 200, 100	

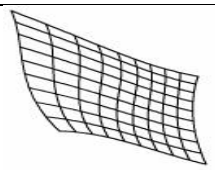
Tabell 4.2 Andre Bezier Flate

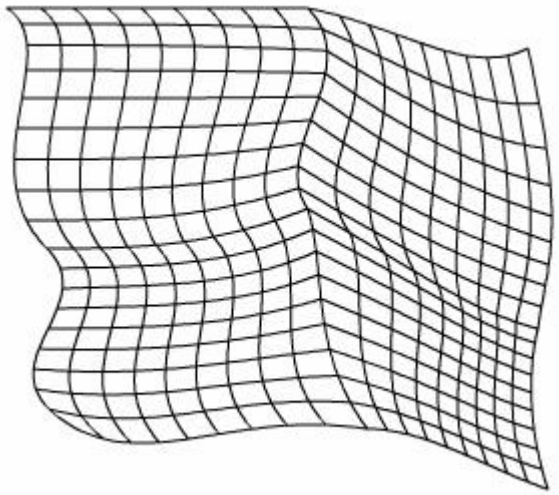
Bezier type	Ankerpunkt1 (x, y, z)	Kontrollpunkt1 (x, y, z)	Kontrollpunkt2 (x, y, z)	Ankerpunkt2 (x, y, z)	Bilde
cubic	70, 220, 200	100, 250, 160	30, 280, 160	85, 310, 180	
cubic	110, 220, 170	130, 245, 170	110, 280, 170	125, 330, 160	
cubic	150, 220, 150	190, 250, 160	130, 280, 160	165, 300, 160	
cubic	200, 200, 100	210, 230, 100	200, 270, 100	230, 310, 100	

Tabell 4.3 Tredje Bezier Flate

Bezier type	Ankerpunkt1 (x, y, z)	Kontrollpunkt1 (x, y, z)	Kontrollpunkt2 (x, y, z)	Ankerpunkt2 (x, y, z)	Bilde
cubic	200, 100, 100	230, 130, 100	180, 170, 100	200, 200, 100	
cubic	250, 110, 100	270, 180, 100	230, 200, 100	270, 240, 100	
cubic	280, 135, 100	290, 185, 100	300, 220, 100	280, 250, 100	
cubic	310, 120, 100	330, 210, 100	320, 225, 100	315, 260, 100	

Tabell 4.4 Fjerde Bezier Flate

Bezier type	Ankerpunkt1 (x, y, z)	Kontrollpunkt1 (x, y, z)	Kontrollpunkt2 (x, y, z)	Ankerpunkt2 (x, y, z)	Bilde
cubic	200, 200, 100	210, 230, 100	200, 270, 100	230, 310, 100	
cubic	270, 240, 100	280, 260, 140	260, 290, 170	260, 310, 160	
cubic	280, 250, 100	290, 270, 120	270, 300, 120	290, 330, 120	
cubic	315, 260, 100	310, 280, 100	310, 310, 100	320, 340, 100	



Figur 4.3 Fire Bezier flater som danner en større flate

Flaten kan tegnes opp på to måter. Enten ved å tegne flatekurve for flatekurve eller ved å tegne rute for rute. Tiden for å tegne rute for rute var betraktelig mye større enn ved flatekurve for flatekurve. Grunnen til dette er at metoden for

opptegning av rute for rute krevde utregning av ulike variabler og ble dermed stor og tung. Tabellen 4.5 viser tidsforskjellen mellom å tegne rute for mot tegning av flatekurve for flatekurve.

Tabell 4.5 Tidsforskjell mellom rute opptegningsmetoder for Bezier flate

Metode	Tid
Flatekurve for flatekurve	930 ms
Rute for rute	2176 ms
Forskjell:	1246 ms

4.2 MATEMATISKE UTTRYKK

En av funksjonene til denne 3D modulen er å kunne ta imot matematiske uttrykk fra den syntetiske kalkulatoren. Siden det var svært usikkert hvor effektivt modulen ville fungere i Flash, ble det kun fokusert på ”hardkodede” uttrykk ble satt som funksjon av y. Per Henrik Hogstad som arbeider med den syntetiske kalkulatoren for parAbel har allerede utviklet en algoritme som analyserer uttrykkene fra kalkulatoren, men dette gjelder kun uttrykk med to ukjente. Arbeid med å videreutvikle denne algoritmen ble lagt på vent, frem til det var påvist at det var mulig å behandle 3D objekter i Flash.

4.3 HYPERBOLSK PARABOL FLATE

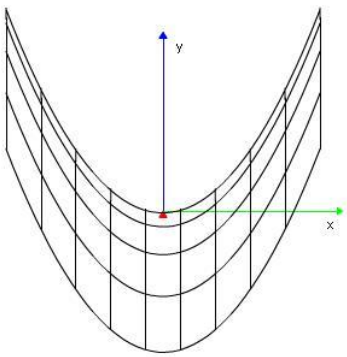
Det viste seg at det var mulig å generere enkle tredimensjonale flater ved hjelp av kubiske Bezier kurver. Det var derfor ønskelig derfor å se på mer avanserte flater generert ut i fra matematiske uttrykk. Valget ble derfor en hyperbolsk parabol. Formelen for en hyperbolsk parabol er:

$$y = \frac{b}{a^2 \cdot x^2} - \frac{b}{c^2 \cdot z^2}$$

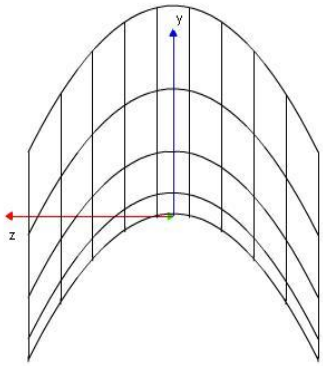
Flaten ble tegnet opp ved å først å la x variere med hensyn på faste z verdier.

Deretter ble z variert med hensyn på faste x verdier. Dette er en rask og effektiv metode for å kalkulere de nødvendige punktene som trengs for å vise flaten. Dessverre ble denne metoden altfor begrenset i forhold til å utføre operasjoner på flaten som fargelegging og normalvektor for hver ”rute”. Dermed er denne metoden kun egnet til å visualisere flaten. Flaten ble derfor generert ved å tegne opp hver ”rute”. Dette viste seg å ha noen negative sider. Metoden for å kalkulere alle punktene ble stor og tung. I tillegg fører denne fremgangsmåten til at de fleste linjene blir tegnet opp to ganger. Kalkulasjonstiden gikk betraktelig opp og Flash brukte tid på å visualisere flaten. Det er selvfølgelig flere faktorer som har innvirkning i forhold til antall kalkulasjoner. Antall ”horisontale” og ”vertikale” kurver øker antall kalkulasjoner samtidig som lengden per linjesegment som blir benyttet i lineTo() metoden også har innvirkning på antall punkter som må kalkuleres.

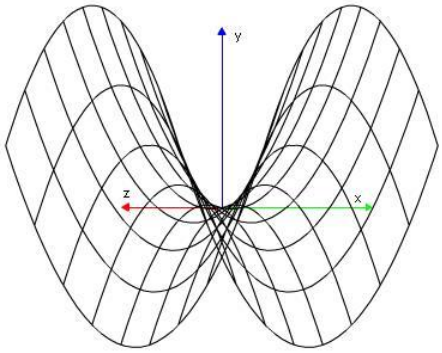
Grensene til den første flaten ble satt fra -130 til +130 for X og fra -130 til +130 i Z. Antall ”horisontale” og ”vertikale” kurver ble satt til ti som standard for testing. Konstantene til x, y og z er satt til a = 10, b = 1 og c = 12. Dette gir flaten en bred bue som gjør den brukervennlig for testing. Størrelsen på linjesegmentene ble satt til en. Dette vil si at en linje for eksempel fra origo til x lik 130 langs x-aksen vil bli dannet ved hjelp av 130 små linjesegmenter. Dette viste seg å ta noe tid, men raskt nok i forhold til å kunne visualisere flaten første gangen. Denne forsinkelsen blir diskutert senere i kapittel 8. Se figurene 4.4, 4.5 og 4.6.



Figur 4.4 En hyperbolsk parabol rotert 0 grader rundt y aksen



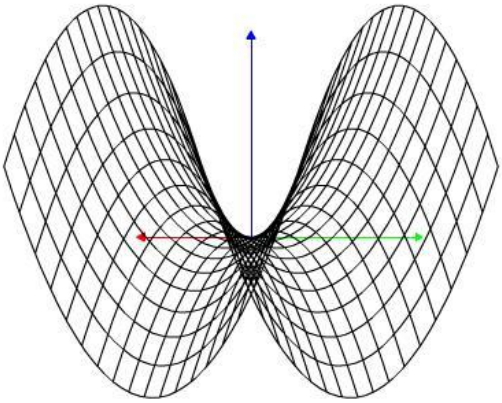
Figur 4.5 En hyperbolsk parabol rotert 90 grader rundt y aksen



Figur 4.6 En hyperbolsk parabol illustrert med 10x10 ruter og rotert 45 grader rundt y aksen

Det er lagt opp til at det skal være mulig å endre antall horisontale og vertikale linjer på figuren. Grunnen til at dette har blitt gjort er både for å kunne endre det visuelle inntrykket, dersom 10x10 skulle vise seg å være misledende i noen tilfeller, og for å kunne utføre en del operasjoner som for eksempel normalvektor på en mer nøyaktig måte. Det bør allikevel eksistere noen begrensinger i forhold til maksimum og minimum antall kurver. Økes linjenummeret i ”vertikal” og ”horisontal” retning til 20x20 dobles antall punkter som må kalkuleres og kan føre til forsinkelser dersom enkelte andre tunge metoder, som fargelegging, skulle kjøres i tillegg. Se figur 4.7. Dette kommer spesielt til syne dersom brukeren skulle velge 100x100 linjer. Dette ville sannsynligvis føre til at hele applikasjonen bryter sammen og må derfor unngås. Et problem som kan oppstå er at hvis det blir lagt inn restriksjon på antall linjer en

figur skal bestå av, finnes det kanskje flater som blir best presentert ved hjelp av mange linjer, men at andre funksjoner som rotasjon og lignende ikke trenger være raskt og effektivt. I tillegg kan selv standard 10x10 linjer virke som for mange linjer i en mindre flate. Dersom grensene er for små vil brukeren sannsynligvis skalere opp flaten slik at den vil få en størrelse som vist på figuren under. Maksimalt størrelse på rutenettet bør derfor settes til 20x20 ruter, slik at andre metoder som rotasjon og lignende ikke skaper store forsinkelser. Det bør allikevel være mulig å overstyre denne grensen, men da må det gis melding til brukeren om at enkelte funksjoner kan ta tid. I andre tilfeller vil det kanskje også være naturlig å endre antall linjer i ”horisontal” retning, men ikke i ”vertikal” retning. Det vil med andre ord være mange muligheter for å stille inn en figur på, men spørsmålet som så dukker opp er brukervennlighet og oversikt.



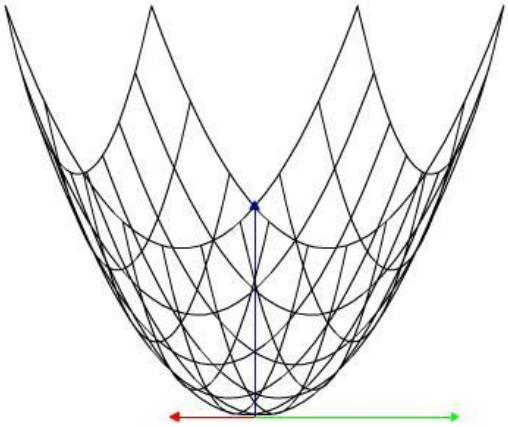
Figur 4.7 En hyperbolsk parabol illustrert med 20x20 ruter og rotert 45 grader rundt y aksen

4.4 ANDRE FLATER

Etter den hyperbolske parabolen var implementert var det ønske om å visualisere andre flater. Dette kunne realiseres på en enkel måte, ved å endre formelen for en hyperbolsk parabol, og danne elliptisk parabol. Formelen for en elliptisk parabol vist under og illustrert på figur 4.8.

$$y = \frac{b}{a^2 \cdot x^2} + \frac{b}{c^2 \cdot z^2}$$

Flaten har samme grenser og konstanter som den hyperbolske parabolen i forrige kapittel.



Figur 4.8 Elliptisk parabol

Andre flater som det er blitt testet på i 3D modulen er avbildet i vedlegg 1.

4.5 PEDAGOGIKK OG FLATER

Flater er et stort kapittel av matematikken og det er ofte vanskelig å se for seg hvordan en flate ser ut bare ved å se på et matematisk uttrykk. Dersom det skal trekkes opp mot undervisning blir det ofte benyttet illustrasjoner fra ulike papirutgaver eller at læreren selv skal prøve å tegne opp flaten for hånd. En ulempe med denne måten er at snittet av flaten som blir presentert ikke nødvendigvis er den beste illustrasjonen for alle studentene. I tillegg kan en håndtegnet figur ha en viss unøyaktighet, som kan være med på å gjøre tolkningen av figuren vanskeligere. Enkelte kalkulatorer i dag har muligheten til å presentere flater, men på grunnlag av størrelse på vinduet i kalkulatoren er det ofte vanskelig å tolke flaten, selv om den her blir presentert korrekt. En bedre løsning vil da være å få flaten presentert korrekt

ved hjelp av en web-applikasjon. Da vil også studenten selv kunne velge hvilket snitt av flaten som gir størst forståelse i en gitt situasjon.

Selv om det å visualisere en flate i seg selv er nyttig, vil det være mulig å benytte disse flatene videre for å visualisere andre matematiske metoder som eksempelvis normalvektor eller tangentplan. Dette er metoder som det ofte kan være vanskelig å se. En løsning er å benytte en flate som de matematiske metodene kan visualiseres på. Dette kan være med på å øke forståelsen raskere.

Ved bruk av Bezier flater er det mulig å generere mindre flater med en spesiell form. Det finnes flere komplekse flater der kun enkelte områder er interessante å ta tak i. Matematikken for en slik del av en flate kan derfor virke mer komplisert, siden hele flaten har blitt presentert. En løsning vil derfor kunne være å generere den delen av flaten som er interessant ved hjelp av Bezier flater og dermed unngå at resten av flaten blir tatt med i betraktning. Dette gjelder selvfølgelig kun dersom resten av flaten ikke har noen form for innvirkning på funksjonen som blir benyttet.

Interaksjon opp mot flater må legges i forhold til tiltenkt læring. I enkelte tilfeller vil det være ønskelig at en bruker kan endre punkter på en flate for å se virkningen av hvordan en tangent endre seg. Er det derimot en gitt flate som skal visualiseres, som for eksempel hyperbolsk parabol, vil det ikke være hensiktsmessig å gi brukeren mulighet til å endre punkter siden dette vil gi feil innformasjon. Et synspunkt kan være å gi brukeren mulighet til stille på noen punkter for å øke forståelsen av flaten, men dersom brukeren ikke stiller tilbake disse punktene eller glemmer å stille tilbake noen av punktene vil det oppstå samme problem med feilinformasjon som diskutert over. Det bør derfor eksistere begrensinger med hensyn til interaksjon i forhold til hva som er ønskelig å formidle.

Oversikt er vel en de argumentene som står sterkest opp mot digital visualisering av matematiske uttrykk. For at denne oversikten skal bli ivaretatt må det derfor eksistere ulike metoder i applikasjonen. Rotering av flaten er et viktig argument. Dette vil føre til at brukeren kan velge det snittet som gir størst forståelse. Det bør også være mulig å skalere flaten i forhold til brukers ønske. Ofte er grensene til en flate små og det må derfor være mulig å gjøre flaten større eller mindre uavhengig av hva grensene til flaten er. Translasjon eller forflyttning kan være nødvendig dersom det eksisterer flere objekter i ”verdenen”. Det kan også være naturlig å flytte bort figuren fra origo, dersom det er ønskelig å ha et opptegnet koordinatsystem ved siden av flaten i stedet for midt i. Dette er diskutert nærmere i

senere kapittel.

Det har til nå blitt diskutert ulike parametere som spiller inn på hvordan en flate skal visualiseres. Eksempler på disse innstillingene er; antall linjer ”horisontalt”, antall linjer ”vertikalt”, antall linjer begge veier, konstanter til x, y og z, detaljrikheten av opptegningen (antall enheter et linjesegment skal gå over), rotasjon i x, y og z, skalering i x, y og z, translasjon i x, y og z eller mulighet for å velge ulike projeksjoner (perspektivisk eller isometrisk). Selv om alle disse muligheten kan være nyttige, kan de også ha en forvirrende effekt. Det er selvfølgelig mulig å begrense mulighetene til et minimum, men det innebærer også en klar begrensning av bruksområde til applikasjonen. En annen mulighet vil være å definere to eller flere grensesnitt, der grensesnittet blir satt etter kunnskapsnivå. Det vil da være mulig å ”låse” opp andre funksjoner dersom det skulle være ønskelig.

5 LAGRING

For at det skal være mulig å rotere, skalere, forflytte eller benytte andre matematiske metoder på en generert flate, må alle punktene til flaten lagres i en eller annen form. Når et objekt blir generert av applikasjonen blir det generert et objekt av klassen som inneholder koden for objektet. Her blir også den nødvendige informasjonen om objektet sendt til klassen som lagrer all informasjon. Denne klassen har fått navnet AllObjects. I denne klassen ligger det en liste som holder på alle objektene som er i bruk. For å unngå at det blir inkonsistens i listen er det viktig at det kun eksisterer en slik liste. Det må derfor benyttes en metode som forsikrer at det kun blir opprettet et objekt av denne listen.

5.1 SINGLETON DESIGN PATTERN

For at det kun skal opprettes et objekt av lagringslisten som er tilgjengelig for alle klasser benyttes Singleton Design Pattern [28]. Denne strukturen går ut på å la klassen og listen være statisk. Samtidig må det testes om listen er blitt generert eller ikke. Dersom listen er blitt generert, benyttes denne listen til lagring, hvis ikke blir først en metode kjørt som oppretter denne listen. Dette er et enkelt system, men løser problemet med flere inkonsistente versjoner av listen.

5.2 LAGRINGSSTRUKTUR

Listen i AllObjects klassen inneholder ett objekt for hver figur som eksisterer i verdenen. Er det blitt tegnet opp et koordinatsystem og en hyperbolsk parabol flate, vil denne listen bestå av fire elementer. Der den visuelle x aksen (grønn pil på figur 5.3) vil være et element. Det samme gjelder y og z aksen. Det siste elementet i listen vil være den hyperbolske parabolen. Alle disse objektene har noen felles parametere. Disse er listet opp i tabell 5.1 nedenfor.

Tabell 5.1 Felles parametere for lagret objekter

Parameter navn:	Beskrivelse:
name	Dette er navnet på objektet. Får et unikt navn dersom det ikke blir definert.
type	Dette identifiserer hvilken type objekt som er lagret. Er denne verdien ”1” vil dette si at objektet er en linje. Verdi ”4” vil si en hyperbolsk parabol uten farger, mens ”5” vil si med farger.
col	Dette er fargen på linjene som blir tegnet. Den har ingen påvirkning på fyllfargene.

Utenom disse er det spesifisert egne parametere for hver type objekt. Dette har blitt gjort for å effektivisere mengden av lagret informasjon til et minimum. En linje vil for eksempel kun ha et startpunkt og et endepunkt i tillegg til felles de parameterne. Det kunne selvfølgelig vært naturlig å bygge videre på denne strukturen for andre objekter og definert punkt tre, punkt fire osv. Problemet kommer når det skal lagres større objekter som den hyperbolske parabolen. Sett at denne har 1000 punkter som skal lagres ville det vært svært tungvindt å lage en metode som automatisk finner alle koordinatene ut i fra parameternavnene til objektet. Det er allikevel bygd videre på denne strukturen opp til et vist antall punkter. Eksempelvis har cubic Bezier kurve lagret alle sine fire punkter som egne punkter i objektet. Under er det vist de ulike parameterne til noen av objektene.

Tabell 5.2 Parameterne til en lagret linje

Linje	
Parameter navn:	**Beskrivelse:**
firstPoint	Er det første punktet til linjen. Er av type Object og holder på parameterne x, y og z.
secPoint	Endepunktet til linjen. Er av samme type objekt som firstPoint

Tabell 5.3 Parameterne til en lagret kubisk Bezier kurve

Kubisk Bezier kurve	
Parameter navn:	**Beskrivelse:**
startPoint	Startpunktet for Bezier kurven. Er av type Objekt og holder på x, y og z verdien til punktet.
controlPoint1	Er det første kontrollpunktet til Bezier kurven. Samme type som startPoint.
controlPoint2	Er det andre kontrollpunktet til Bezier kurven. Samme type som startPoint.
endPoint	Endepunktet til Bezier kurven. Samme type som startPoint.

Tabell 5.4 Parameterne til en lagret kubisk Bezier flate

Kubisk Bezier flate	
Parameter navn:	**Beskrivelse:**
curve01	Den første bærekurven for flaten. Er av type Array og holder på fire 3D-punkter av type Object.
curve02	Den andre bærekurven for flaten. Samme type som curve01.
curve03	Den tredje bærekurven for flaten. Samme type som curve01.
curve04	Den fjerde bærekurven for flaten. Samme type som curve01.

Tabell 5.5 Parameterne til en hyperbolsk parabol flate

Hyperbolsk parabol	
Parameter navn:	**Beskrivelse:**
coords	Koordinatene til flaten, sammen annen informasjon, som når hver rute starter og slutter. Består av en multidimensjonal Array, som er beskrevet i tabellen nedenfor.
filled	Definerer om flaten er fylt med farge. Har verdier fra 0, 1, 2 osv. Der ”0” definerer ingen fyllfarge, ”1” definerer fylt med enkel farge i hver rute, ”2” definerer fylt med gradientfarge i hver rute.

For å kunne behandle den hyperbolske parabolen etter den har blitt generert, må alle punktene ha blitt lagret sammen med informasjon om når Flash skal begynne å tegne og når den kun skal flytte tegnepunktet til et annet sted.. Dersom det kun skal genereres en linjeformet flate, er dette all informasjon som er nødvendig. Det vil

da bli laget en multidimensjonal array bestående av to rader. Når det blir generert en multidimensjonal array i ActionScript definerer den antall rader i stedet for antall kolonner. Skal en lagre en hyperbolsk parabol med farge, trengs det tilleggsinformasjon om når Flash skal begynne å fylle, slutte å fylle og hvilke(n) farge ruten skal fylles med. Det er også lagret informasjon om hvor hjørnene til rutene er, slik at det skal være enkelt å hente ut en spesifikk rute. Strukturen til arrayen er vist i tabell 5.6 nedenfor.

Tabell 5.6 Koordinat listen til en hyperbolsk parabol

Koordinat listen til en hyperbolsk parabol	
Rad:	**Beskrivelse**
[0] l/m	Denne raden definerer om Flash skal tegne en linje eller flytte tegnemarkøren. ”l” definerer bruk av metoden lineTo() og impliserer at det skal tegnes en linje til punktet i neste rad. ”m” definerer bruk av metoden moveTo() og impliserer at tegnemarkøren skal flyttes.
[1] Object(x,y,z)	Denne raden inneholder alle punktene til flaten. Punktene er av type Object og holder på x, y og z koordinatene.
[2] farger	Denne raden inneholder informasjon om hvilke(n) farger som hver rute skal ha. Er definert i starten av hver rute.
[3] begin/end	Innholder informasjon om når fyllingen av en rute starter og slutter. Inneholder også informasjon om hvor hjørnepunktene er. ”begin” vil definere start av fargefylling og første hjørnepunkt. ”corner” definer de tre andre hjørnepunktene og ”end” definerer når fargefyllingen slutter.

5.3 BRUK AV EKSTERNE PUNKTSETT

Det kan ofte være ønskelig å ha definert punktsettet til en kurve i eksterne medier. Selv om vår løsning har benyttet bruk av array lister til lagring, vil det være mulig å benytte eksterne punktsett lagret i egne filer.

5.4 DISKUSJON

Punktsett som er lagret eksternt må håndtere noen krav. Flash sin lineTo() metode tegner en linje til et gitt punkt fra forrige punkt som er tegnet opp. Dersom det ikke er tegnet opp noe, går Flash ut i fra punktet (0,0). Dette vil si at dersom det ikke

foreligger informasjon om at tegnemarkøren skal flyttes vil det oppstå uønskede linjer mellom punkter som ikke skal ha kontakt. Et annet aspekt er hurtigheten. Rotering av en flate eller lignende medfører endring av koordinatene til flaten. Lesing og skriving til og fra en fil tar tid, slik at det kan være mer hensiktsmessig å kopiere punktene inn i programmet og gjøre endringene der i stedet.

6 TRANSFORMASJON

Behovet for rotasjon i en 3D modul er stort. Dette for å se objektet i flere mulige posisjoner for å få innsyn i oppbygningen av et 3D objekt, og få en virkelighetsnær opplevelse av formen til objektet. I dette kapittelet blir det gjennomgått to forskjellige mattematiske metoder for rotasjon i den 3-dimesjonale verden. De to metodene er matriserotasjon og quaternionrotasjon. Metodene og mattematikken bak metodene vil bli beskrevet. Matematikken som er brukt i dette kapittelet er hentet fra forskjellige mattematikklærebøker, samt internett [6] [29] [30] [31]

6.1 MATRISE

6.1.1 Matematikken

Innenfor matematikken er en matrise beskrevet som en rektangulær tabell bestående av tall. Man kan også si at det er en tabell som består av abstrakte mengder som man kan utføre operasjoner på, som addering, multiplisering, mm.

Et eksempel på en 4×3 matrise vises i figur 6.1 under:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 7 \\ 4 & 9 & 2 \\ 6 & 0 & 1 \end{bmatrix}$$

Figur 6.1 En 4x3 matrise

De horisontale linjene heter rekker, og de vertikale linjene kolonner. Matrisen over kalles en 4×3 matrise fordi det er 4 rekker og 3 kolonner, og navngis alltid i denne rekkefølgen, rekker×kolonner.

Addering av Matriser

Anta at to m×n matriser skal adderes, matrise A og matrise B. Summen av disse matrisene(A + B) gir matrise C, som også blir en m×n. Hvert element i matrise A blir lagt til tallet i samme posisjon i matrise B, og de tallene addert blir det nye tallet som skal på lik posisjon i matrise C.

Nedenfor, figur 6.2, er det vist et eksempel på en addisjon av A + B = C.

A B A + B C

$$\begin{bmatrix} 1 & 2 & 2 \\ 2 & 0 & 0 \\ 1 & 2 & 2 \end{bmatrix} + \begin{bmatrix} 2 & 0 & 5 \\ 5 & 2 & 0 \\ 2 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1+2 & 2+0 & 2+5 \\ 2+5 & 0+2 & 0+0 \\ 1+2 & 2+1 & 2+1 \end{bmatrix} = \begin{bmatrix} 3 & 2 & 7 \\ 7 & 2 & 0 \\ 3 & 3 & 3 \end{bmatrix}$$

Figur 6.2 Addisjon av to matriser A og B som gir matrisen C

Multiplisering av matriser

Anta at to matriser skal multipliseres, matrise A og matrise B. Resultatmatrisen av dette produktet kalles C.

For å kunne multiplisere to matriser er det ett kriterium som må være oppfylt. Antallet kolonner i den første matrisen(A) må være likt antall av rekker i den andre matrisen(B). Siden dette kriteriet må være oppfylt, vil resultatmatrisen få forskjellige antall rader og kolonner i forhold til matrisene som blir multiplisert. Anta at A er en m×n matrise, og B er en n×o matrise. Resultatmatrisen, C, av A × B vil da bli en m×o matrise. Det vil si at matrise C vil ha m antall rekker og o antall kolonner. Se figur 6.3. Matrisemultiplikasjon er ikke kommutativ. Det vil si at AB er ikke lik BA($AB \neq BA$).

A B

$$\begin{bmatrix} 1 & 1 & 2 \\ -1 & 3 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 2 & 1 \\ 1 & 3 \end{bmatrix}$$

A x B C

$$= \begin{bmatrix} (1\times 0+1\times 2+2\times 1) & (1\times 1+1\times 1+2\times 3) \\ (-1\times 0+3\times 2+0\times 1) & (-1\times 1+3\times 1+0\times 3) \end{bmatrix} = \begin{bmatrix} 4 & 8 \\ 6 & 2 \end{bmatrix}$$

Figur 6.3 Multiplikasjon av to matriser A og B som gir resultat matrisen C

Det finnes også en del flere regneregler for matrisemultiplikasjon. Her er de mest vanlige:

(AB)C = A(BC), matrise A (m×n), matrise B(n×o), matrise C(o×p)
(A + B)C = AC + BC), matrise A/B (m×n), matrise C(n×o)
C(A + B) = CA + CB), matrise A/B (m×n), matrise C(o×m)

Skalar multiplikasjon

Skalar multiplikasjon med matrise går mye ut på det samme, bare her multipliseres et tall med en matrise. Da blir hvert tall i en matrise A multiplisert med et fast tall.

Se figur 6.4 under:

A 2 x A C

$$2\begin{bmatrix} 1 & 2 & -3 \\ 3 & -2 & 1 \end{bmatrix} = \begin{bmatrix} 2\times 1 & 2\times 2 & 2\times -3 \\ 2\times 3 & 2\times -2 & 2\times 1 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -6 \\ 6 & -4 & 2 \end{bmatrix}$$

Figur 6.4 Skalar multiplikasjon av en variabel med en matrise A som gir resultat matrisen C

6.1.2 Bruk av matematikken i oppgaven

Rotasjon

Innenfor mattematikkens verden finnes det satte matriser for å rotere i hver enkel retning. Det vil si at det er en matrise for rotasjon rundt x-aksen, en rundt y-aksen og en rundt z-aksen. Disse matrisene inneholder informasjon om hvor mange grader(θ) som objektet skal rotere rundt den valgte aksen. Hvilken vei objektet skal rotere rundt aksen, kommer an på fortegnene til tallene i matrisen. Nedenfor kan man se eksempel, figur 6.5, 6.6 og 6.7, på rotasjonsmatriser som går med klokken, sett fra positiv side:

$$\mathcal{R}(\alpha) := \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{pmatrix}$$

Figur 6.5 Rotasjon rundt x-aksen

$$\mathcal{P}(\gamma) := \begin{pmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{pmatrix}$$

Figur 6.6 Rotasjon rundt y-aksen

$$\mathcal{Y}(\beta) := \begin{pmatrix} \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Figur 6.7 Rotasjon rundt z-aksen

Det finnes også en mer sammensatt matrise som samler alle disse operasjonene innefor kun denne ene matrisen. Denne matrisen er vist nedenfor

i figur 6.8.

$$M(\mathbf{v},\theta) =$$

$$\begin{bmatrix} \cos\theta + (1-\cos\theta)x^2 & (1-\cos\theta)xy - (\sin\theta)z & (1-\cos\theta)xz + (\sin\theta)y \\ (1-\cos\theta)yx + (\sin\theta)z & \cos\theta + (1-\cos\theta)y^2 & (1-\cos\theta)yz - (\sin\theta)x \\ (1-\cos\theta)zx - (\sin\theta)y & (1-\cos\theta)zy + (\sin\theta)x & \cos\theta + (1-\cos\theta)z^2 \end{bmatrix}$$

Figur 6.8 Sammen satt matrise med mulighet for rotasjon i alle retninger

For å få hele objektet til å rotere som ønsket, må hvert punkt tilhørende objektet multipliseres med rotasjonsmatrisen. Desto flere punkter objektet består av, vil det bli flere kalkulasjoner som øker prosessortiden for utregninger av rotasjonen. Hvert punkt har en x, y og z verdi. De nye verdiene etter multipliseringen med rotasjonsmatrisen vil være x', y' og z'. I figur 6.9 som er vist under vil punktet bli rotert rundt en vektor v = (i, j, k), som kan ha verdier 0 eller 1, alt etter som hvilken akse punktet skal rotere om. Det vil også være en verdi θ, som sier hvor mange grader punktet skal rotere.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta + (1-\cos\theta)x^2 & (1-\cos\theta)xy - (\sin\theta)z & (1-\cos\theta)xz + (\sin\theta)y \\ (1-\cos\theta)yx + (\sin\theta)z & \cos\theta + (1-\cos\theta)y^2 & (1-\cos\theta)yz - (\sin\theta)x \\ (1-\cos\theta)zx - (\sin\theta)y & (1-\cos\theta)zy + (\sin\theta)x & \cos\theta + (1-\cos\theta)z^2 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix}$$

Figur 6.9 Bruk av den sammensatte rotasjonsmatrisen for rotering av et gitt punkt

Forflyttning

Hvert objekt som blir visualisert i 3D modulen blir laget i koordinatsystemets origo. Skal det lages flere 3D objekter, må det finnes muligheter for å flytte objektene slik at de ikke overlapper hverandre og skaper uorden. Det er en av grunnene til at 3D modulen bør ha en mulighet til translasjon av objekter. Flytter man et objekt fra objektets originale punkter, vil ligningen til objektet i teorien også forandre seg. Man kan kompensere for

dette ved hjelp av et internt koordinatsystem til hvert objekt, eller bruke det til å se forandringene i ligningen med tanke på læring.

Det finnes flere måter å utføre translasjon av punkter på. En mulighet er translasjon ved hjelp av matriser, der en translasjonsmatrise blir multiplisert med punktmatrisen. Matriser med homogene koordinater blir brukt, siden forflyttning ikke er mulig med 3 × 3 matriser. Multiplikasjonen er kommutativ i motsetning til vanlig matrise multiplikasjon. Dette fordi addering av vektorer er kommutativt. Et eksempel på denne metoden er vist i figur 6.10.

$$\mathbf{F_v p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} p_x + v_x \\ p_y + v_y \\ p_z + v_z \\ 1 \end{bmatrix}$$

Figur 6.10 Forflytning av punkter ved hjelp av en homogen matrise

En enklere versjon for flytting av objekter ved hjelp av matriser, er å la matrisen til punktet(p) addert med en forflyttningsmatrise(v) uten bruk av homogene koordinater. Et eksempel er vist under i figur 6.11:

$$\mathbf{p} + \mathbf{v} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} p_x + v_x \\ p_y + v_y \\ p_z + v_z \end{bmatrix}$$

Figur 6.11 Forflytning av punkter uten bruk av en homogen matrise

Bruker man ikke matriser i utviklingen av 3D modulen, kan man forflytte objekter ved hjelp av en enkel metode, mye lik siste eksempel(uten bruk av matriser). Denne metoden går ut på at P_x, P_y og P_z koordinatene til punktet blir addert med en verdi V_x, V_y og V_z i hver retning for å få forflyttning som ønskes.

Skalering

Det er flere faktorer som spiller inn i skalering av objekter i en 3-dimensjonal verden. Skaleringsfaktoren kan være lik i alle retninger(x, y og z), eller objekter kan skaleres kun i en eller to retninger. Et eksempel kan være en kule som blir skalert i x retning med en gitt faktor. Resultatet vil da bli en sfæroide, som kan betegnes som en 3D ellipse.

Det blir ofte brukt homogene koordinater når matriser blir brukt i skalering(4 × 4 matriser), men behøves ikke selv om skaleringsfaktoren ikke er lik i alle retninger. Eneste grunnene til at man bruker homogene koordinater ved skalering, er fordi translasjon ikke er mulig med 3 × 3 matriser. Det fører til at punktene til objektet har fire(x, y, z, 1) verdier i stedet for tre(x, y, z), og må da også ha homogene koordinater ved skalering.

Er skaleringsfaktoren lik, kan skaleringen enkel gjøres ved å bruke skalar multiplikasjon nevnt tidligere i kapittelet. Det vi si at koordinatene til hvert punkt vil bli multiplisert med et tall. Et eksempel på en skalering på 100 % vil bli vist nedenfor i figur 6.12:

$$S_v\mathbf{p} = 2\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} 2\,p_x \\ 2\,p_y \\ 2\,p_z \end{bmatrix}$$

Figur 6.12 Et punkt blir skalert 100% større

Finnes det behov for å skalere med ulik faktor i hver retning, må man bruker en skaleringsmatrise som må multipliseres med alle punktene til objektet. Et eksempel på denne type skalering uten homogene koordinater er vist nedenfor i figur 6.13:

$$S_v\mathbf{p} = \begin{bmatrix} v_x & 0 & 0 \\ 0 & v_y & 0 \\ 0 & 0 & v_z \end{bmatrix}\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} v_x p_x \\ v_y p_y \\ v_z p_z \end{bmatrix}$$

Figur 6.13 Skalering av ett punkt ved hjelp av en ikke homogen skaleringsmatrise

Det finnes også en simpel metode der man unngår å bruke matriser ved skalering av et objekt. Metoden er veldig lik matrise multipliserings skalering vist tidligere i kapittelet. Denne metoden går ut på at P_x, P_y og P_z koordinatene til punktet blir multiplisert med skaleringsverdiene V_x, V_y og V_z for å få skaleringen som ønskes i hver av retningene.

6.2 QUATERNION

6.2.1 Matematikk

Innenfor matematikken er quaternions beskrevet som ikke-komuttativ forlengelse av komplekse tall. Komplekse tall oppnås ved at en imaginær enhet, i, er tilsatt et reelt tall. For å oppnå dette må i ^2= −1 tilfredstilles. [31]

En quaternion er teknisk sett fire tall, tre av dem inneholder imaginære enheter. En quaternion er definert på denne måten, q = w + xi + yj + zk, der w, x, y, og z er reelle tall.

Alle de imaginære tallene må tilfredsstille denne ligningen:

$$i^2 = j^2 = k^2 = ijk = -1$$

Ved litt utregninger kan denne ligningen omformuleres til en rekke andre ligninger som viser seg nyttige i forståelsen av imaginære tall, og bruken av disse til utregninger av likninger:

$$\begin{array}{lll} ij = k, & & ji = -k, \\ jk = i, & & kj = -i, \\ ki = j, & & ik = -j. \end{array}$$

Det finnes muligheter for å utrykke quaternions både i vektorbaserte og matrisebaserte operasjoner.

Dette er et eksempel på to quaternion utrykt ved hjelp av vektor:

$$p = a + \vec{u} = a + bi + cj + dk$$

Nedenfor er det et eksempel, vist i figur 6.14 og 6.15, på en quaternion utrykt ved hjelp av matriser. Det finnes to forskjellige måter å gjøre det på. Den ene metoden er å bruke en 2×2 kompleks matrise, den andre er ved å bruke en 4×4 reell matrise. Under vises bruken av begge mulighetene. Quaternion: a + bi + cj + dk

$$\begin{pmatrix} a+bi & c+di \\ -c+di & a-bi \end{pmatrix}$$

Figur 6.14 Quaternion utrykt ved hjelp av en 2x2 kompleks matrise

$$\begin{pmatrix} a & -b & d & -c \\ b & a & -c & -d \\ -d & c & a & -b \\ c & d & b & a \end{pmatrix}$$

Figur 6.15 Quaternion utrykt ved hjelp av en 4x4 reell matrise

Quaternion multiplikasjon

Quaternion er vektorbasert, og har derfor samme multiplikasjons utregninger. Nedenfor er det vist eksempler der to quaternion, p og q, blir multiplisert:

$$p = a + \vec{u} = a + bi + cj + dk$$
$$q = t + \vec{v} = t + xi + yj + zk$$

Figur 6.16 Definisjon av de to quaternion, p og q

Den vanligste måten å multiplisere(ikke-komuttativ) to quaternion på kalles Grassmann produktet. Den er beskrevet i figur 6.17:

$$pq = at - \vec{u} \cdot \vec{v} + a\vec{v} + t\vec{u} + \vec{u} \times \vec{v}$$

Figur 6.17 Multiplikasjon med to quaternion ved hjelp av Grassmann produktet

Siden det er en ikke-komuttativ multiplikasjon av to quaternioner, vil ikke resultatet av pq være det samme som qp. Dermed blir også Grassmann produkt brukt i løsningen av ligningen nedenfor.

Nedenfor er produktet av q multiplisert med p beskrevet i figur 6.18:

$$qp = at - \vec{u} \cdot \vec{v} + a\vec{v} + t\vec{u} - \vec{u} \times \vec{v}$$

Figur 6.18 Produktet av q multiplisert med p ved hjelp av Grassmann produktet

Det finnes flere metoder brukt til å regne ut multiplikasjoner mellom flere quaternion. En av metodene heter Euclidean produktet, se figur 6.19. I stedet for å bruke den første quaternionen blir den konjugerte av denne brukt. Et eksempel, vist under, på den konjugerte:

$o = a + ib$

$\bar{o} = a - ib$

$o \cdot \bar{o} = a^2 + b^2$

$$p^*q = at + \vec{u} \cdot \vec{v} + a\vec{v} - t\vec{u} - \vec{u} \times \vec{v}$$

Figur 6.19 Eksempel på bruk av Euclidean produktet

På samme måte som Grassmann produktet er Euclidean produktet også en ikke-komuttativ multiplikasjon. Dette vil si at resultatet p*q ikke d samme som q*p.

Nedenfor, i figur 6.20, er produktet av q multiplisert med p beskrevet:

$$q^*p = at + \vec{u} \cdot \vec{v} - a\vec{v} + t\vec{u} + \vec{u} \times \vec{v}$$

Figur 6.20 Produktet av to quaternion, p og q, ved hjelp av Euclidean produktet

6.2.2 Bruk av mattematikken i oppgaven

Quaternions er fire dimensjonale objekter bestående av x, y, z og w som beskriver akserotasjon(hvilke akse(r) objektet skal rotere om) og hvor mange grader punktet skal roteres. Slik kan en quaternion se ut: q = w + xi + yj + zk.

Quaternion er ofte brukt i 3D motorer til å raskt rotere punkter i rommet. Hovedgrunnen til at quaternion blir mye brukt i 3D motorer er at det bare eksisterer fire variabler, og dermed blir kravet til minne mindre. Det blir samt mindre kalkulasjoner sammenlignet med matrise metoden for rotasjon av punkter.

Gjennomgang av matrisemetoden kommer i et eget kapittel.

En quaternion lagrer en akse og rotasjonen den skal ha rundt aksen. For å bruke Quaternion rotasjon, må 3D punkter konverteres til quaternion notasjon. Dette gjøres ved å sette x, y og z komponentene til en quaternion lik x, y og z til punktet, og sette w komponenten til null. Når punktet er konvertert er d mulig å multiplisere det med en quaternion for å rotere punktet i den 3-dimensjonale verden. Dette vil bli forklart mer detaljert under.

For å forhindre mattematiske feil ved bruk av desimal tall, må man normalisere quaternionen. Dette fordi nøyaktigheten på desimal tall vil etter hvert minske, og det burde finnes en mulighet til å normalisere quaternionen for igjen å operere med riktige tall. Denne prosedyren er omtrent like enkel som å normalisere en vektor. Lengden av en enhetsvektor blir ofte kalt høyden til vektoren.

Høyden til en quaternion er gitt ved formelen:

$$m = \sqrt{(a*a + b*b + c*c)}$$

For en quaternion skal høyden være 1(m = 1). Høyden blir kontrollert, og dersom den er tilnærmet lik 1, altså innenfor et definert toleranseområde, trengs det ikke en normalisering. Hvis høyden derimot er utenfor toleranseområdet vil quaternion bli normalisert. Punktene etter normaliseringen vil da bli brukt videre i

beregningen av nye rotasjoner.

Variablene a, b og c er x, y og z komponentene som bestemmer om hvilke akse punktet blir rotert, og d er graden punktet skal rotere. Rotasjonsgraden er gitt i radianer. Variablene x`, y`, z` og w` er komponentene av den resulterende quaternionen.

Normaliseringen vil skje på denne måten: [32]

```
ca  = cos(d/2);
sa  = sin(d/2);
m  = √ (a*a + b*b + c*c);
x`  = a/m * sa;
y`  = b/m * sa;
z`  = c/m * sa;
w` = ca;
```

Normaliseringen av punktene skal ikke brukes hvis det ikke er nødvendig, mye siden de bruker mye CPU tid og derfor burde unngås.

Skulle det være behov for å kombinere flere rotasjoner så finnes det mulighet for det. Ved å multiplisere flere quaternion, vil det bli laget en kombinasjon av aksene og rotasjonen til hver av dem. Ved å gjøre det på denne måten vil det bli spart prosessorkraft ved å fjerne behovet for å rotere punktene separat med hver sin quaternion.

Her er likningen for hvordan multiplikasjonen er gjennomført: [32]

Q_2*Q_1:

$(Q_1 * Q_2).\$w = w1*w2 - x1*x2 - y1*y2 - z1*z2$

$(Q_1 * Q_2).\$x = w1*x2 + x1*w2 + y1*z2 - z1*y2$

$(Q_1 * Q_2).\$y = w1*y2 + y1*w2 + z1*x2 - x1*z2$

$(Q_1 * Q_2).\$z = w1*z2 + z1*w2 + x1*y2 - y1*x2$

Som beskrevet i mattematikken bak quatenion, skrevet tidligere i kapittelet, er det viktig å huske på at dette er en ikke-komuttativ multiplikasjon. Det vil resultere i at Q_2*Q_1 ikke vil være det samme som Q_1*Q_2.

6.3 DISKUSJON

Det har blitt gjennomgått to ulike matematiske rotasjonsmuligheter for 3D objekter. Hvilken av disse passer best til 3D modulen med tanke på egenskaper og effektivitet?

Matriser kan bli brukt til å representere en rekke nyttige transformasjoner, slik som rotasjon, skalering og translasjon. Blir matriser brukt i en av translasjonene, bør det brukes matriser i alle translasjonene. Da vil det gjerne bli brukt homogene koordinater, altså 4×4 matriser. Denne måten å løse problemet på gjør det mulig å kombinere transformasjoner ved å multiplisere de med hverandre og multiplisere transformasjoner med punktene til et objekt. Hvert punkt til objektet vil ved bruk av homogene koordinater få fire verdier x, y, z og 1. Det er ikke mulig å multiplisere matriser hvis ikke alle transformasjonene har homogene koordinater, og translasjon er heller ikke mulig med 3 × 3 matriser.

Det finnes flere matematiske metoder man kan bruke for å flytte på et punkt i 3D. Som nevnt ovenfor finnes muligheten til å bruke en 4×4 matrise, med homogene koordinater. En annen mulighet er å ikke bruke matrise, men addere en gitt verdi til hver av objektets punkter x, y og z. Denne metoden er svært enkel, og krever færre kalkulasjoner enn bruk av matrisen med homogene koordinater multiplisert med hvert punkt til objektet. Det finnes også en tredje mulighet der man bruker to 3×1 matriser, som ligner på metoden uten bruk av matriser. Denne måten å løse translasjonen på krever like mange kalkulasjoner som metoden uten bruk av matriser, og er dermed ikke mer effektiv.

Skalering av objekter kan utføres på en rekke forskjellige måter. Det kan skaleres med samme verdi i alle retninger, slik at det blir litt mer lik en zoom funksjon, eller skaleres med ulik verdi i retningene. Den siste muligheten vil gjøre det mulig å forandre formen til objektet hvis det er ønskelig. Mattematisk er det flere måter å løse skalering på. Som for translasjon er det også mulig å bruke matriser. Skaleringsmatrisen kan være en 4×4 matrise med homogene koordinater, men kan også være en 3×3 matrise. Jo større matrisene er, desto flere kalkulasjoner må til for å multiplisere matrisene. Disse to metodene er veldig like og gir muligheten for å skalere med ulik verdi i hver retning. Eneste forskjellen på matrisene er bruken av homogene koordinater.

Er det ønskelig å ha en zoom lignende løsning for skalering finnes det

mulighet for dette. Det kan gjøres ved hjelp av skalar multiplikasjon der 3×1 matrisen til punktet blir multiplisert med et gitt tall. Denne metoden er enkel i forhold til de andre metodene, og det finnes bare mulighet for samme skalering i alle retninger. En annen måte, som er den enkleste måten å løse skaleringsproblemet på er å multiplisere x, y og z verdien til punktene med enten en fast eller ulik verdi for hver akse. På denne måten har du alle mulighetene tilgjengelig i tillegg til at dette er den metoden som bruker færrest kalkulasjoner.

Skal det være mulig for rotasjon i 3D modulen finnes det flere metoder å velge mellom. To av disse er matrise rotasjon og quaternion rotasjon. Innenfor 3D verdenen blir quaternion mye brukt, mye fordi det trengs færre kalkulasjoner, og vil derfor være raskere ved rotasjon av mange 3D objekter, eller tunge 3D modeller. Ved bruk av quaternion vil gimbal lock bli forhindret.[33] Quaternion har også muligheten til å kombinere flere rotasjoner før multiplisering med punktet, i motsetning til matrise rotasjon.

Innefor 3D rotasjon ved hjelp av matriser er det definert en rotasjonsmatrise rundt x, en rundt y og en rundt z. Det finnes også en matrise som kombinerer alle disse matrisene. Hvilken av disse bør brukes til 3D modulen, skal det roters rundt kun en akse eller rundt alle samtidig? Det burde finnes mulighet for begge. Basert på dette ville bruken av matrisen som hadde alle rotasjonene krevd minst kalkulasjoner. Det ville også blitt mindre kode ved bruk av en matrise i stedet for flere.

For å bevise at bruken av quaternion er en raskere metode enn bruk av matriser, er det gjennomført tester på en hyperparabol flate bestående av 2720 punkter. Resultatet ble åpenlyst selv på testing av kun et objekt. Årsaken til at quaternion er raskere, er at det trengs færre kalkulasjoner for utregningen av de nye roterte punktene. For å øke nøyaktigheten på forsøket har det blitt tatt ti forsøk med hver metode, og funnet gjennomsnittet. Quaternion metoden brukte nesten et halvt sekund(0,431) mindre enn Matrise metoden. Se tabell 6.1 for oversikt over forsøket.

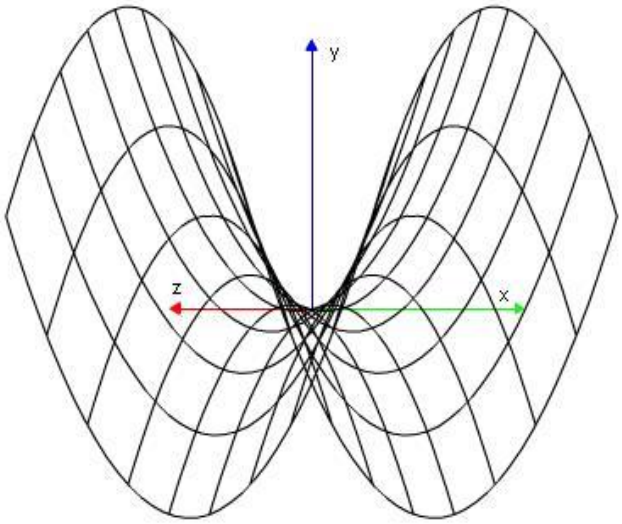
Tabell 6.1 Resultater av rotasjon av hyperparabol med 2720 punkter

Matriserotasjon	*Quaternionrotasjon*
Start 13688 ms Stopp 16393 ms Sum 2705 ms	Start 35798 ms Stopp 38119 ms Sum 2321 ms
Start 17240 ms Stopp 19943 ms Sum 2703 ms	Start 40029 ms Stopp 42338 ms Sum 2309 ms
Start 20787 ms Stopp 23504 ms Sum 2717 ms	Start 49701 ms Stopp 51976 ms Sum 2275 ms
Start 27670 ms Stopp 30381 ms Sum 2711 ms	Start 56750 ms Stopp 59028 ms Sum 2278 ms
Start 34951 ms Stopp 37660 ms Sum 2709 ms	Start 61330 ms Stopp 63617 ms Sum 2287 ms
Start 40231 ms Stopp 42944 ms Sum 2713 ms	Start 68510 ms Stopp 70765 ms Sum 2255 ms
Start 43789 ms Stopp 46499 ms Sum 2710 ms	Start 72802 ms Stopp 75071 ms Sum 2269 ms
Start 48327 ms Stopp 51031 ms Sum 2704 ms	Start 76370 ms Stopp 78629 ms Sum 2259 ms
Start 57619 ms Stopp 60328 ms Sum 2709 ms	Start 79750 ms Stopp 82033 ms Sum 2283 ms
Start 61169 ms Stopp 63904 ms Sum 2735 ms	Start 84554 ms Stopp 86825 ms Sum 2271 ms
Gjennomsnitt 2712 ms	*Gjennomsnitt 2281 ms*

7 FARGER

7.1 RUTENETT

Flater kan representeres ved hjelp av ulike virkemidler. Den enkleste formen er å presentere flaten kun ved hjelp av et rutenett. Dette rutenettet kan igjen tegnes opp på to forskjellige måter. Den mest effektive måten er ved først å tegne alle ”vertikale” linjer for deretter å tegne opp de ”horisontale” linjene. Se figur 7.1. Problemet med denne type opptegning er effektiviteten for å finne en spesifikk rute. Denne egner seg derfor ikke dersom det skal utføres matematiske metoder som krever kunnskap om hver rute på flaten. Det andre alternativet er å tegne opp rute for rute frem til flaten er fylt opp. Dette er en mye tyngre metode og vil i tillegg føre til at flere av linjene blir tegnet opp to ganger. Det positive er at applikasjonen nå har full kontroll på hvor hver enkel rute befinner seg, som gjør det raskt å beregne egenskapene til en rute. Denne type opptegning er også en forutsetning for at en flate kan fylles med farge. Grunnen til dette er at beginFill() metoden til ActionScript krever et lukket område som må avsluttes med endFill() metoden for å fylle med farger. Den tar ikke hensyn til tidligere tegnete linjer, og dette krever at flere av linjene må tegnes opp dobbelt.



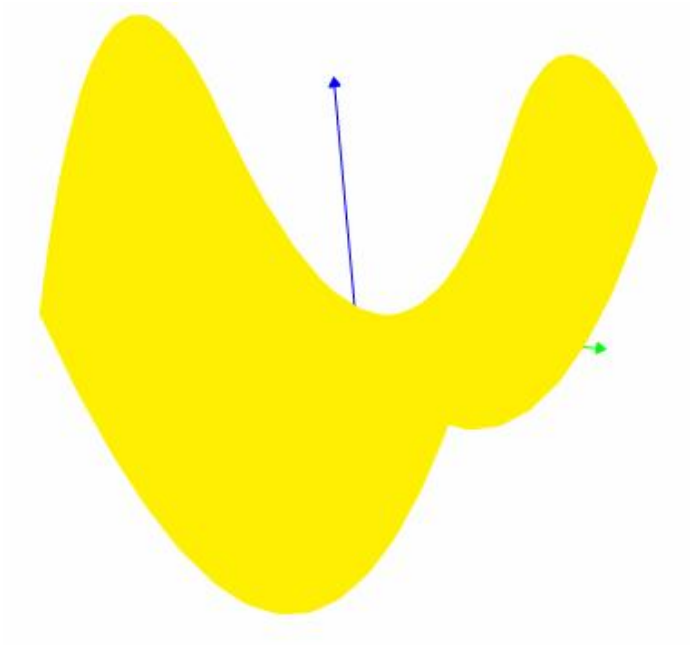
Figur 7.1 Flate representert med rutenett

7.2 FYLT FLATE

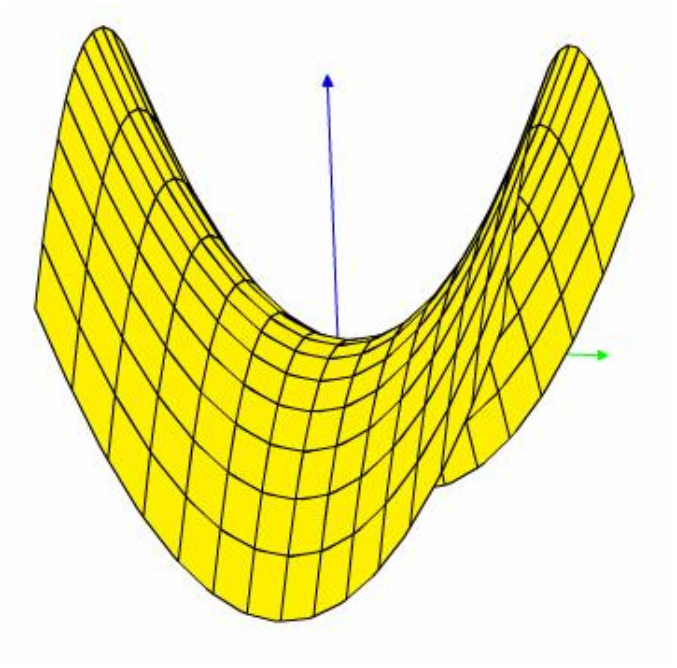
Selv om en linjeformet flate gir en god oversikt over hvordan kurvene som representerer flater er bygd opp, er det mulig å benytte fargelegging av flaten for å oppnå andre visualiseringseffekter. Ulike farger kan representere ulike egenskaper til en flate og det er mulig å benytte ulike fargemønster for å skille de ulike egenskapene.

7.2.1 Fylt flate med en farge

Den enkleste måten å fargelegge en flate på, er å benytte en enkelt farge. Fordelen med denne typen fargelegging er at flaten gir et inntrykk av helhet. En flate bestående av et rutenett kan ha en forvirrende effekt av hvilke linjer som tilknytning til hverandre, dersom rutene overlapper hverandre. Her kan en farget flate både ha en negativ effekt og en positiv effekt. Dersom det ikke blir benyttet et synlig rutenett i tillegg til fyllfargen vil hele flaten få et ”flatt” perspektiv (Figur 7.2). Mye av konturene som gir et romlig bilde vil forsvinne og det vil være svært vanskelig å tolke hvordan flaten krummer. Med et rutenett i tillegg til fyllfargen, vil kun linjene til de synlige flatedelene komme frem. Dette fører til at flaten virker mer sammenhengende og gir en større forståelse av hvilke deler av flaten som befinner seg fremst eller bakerst i en gitt posisjon. En flate representert med en farge og rutenett er vist på figur 7.3 Denne typen bruk av farger har også negative sider. Problemet med ”flatt” perspektiv er nevnt ovenfor, selv om den har mindre negativ effekt med bruk av rutenett. En annen negativ effekt ved bruk av enkel farge er at ujevnheter/ egenskaper ved flaten kan bli gjemt av fargen. Dersom en flate har en mindre høyde mot midten, kan det være vanskelig å legge merke til at denne høyden eksisterer, siden alle områder rundt denne høyden har samme farge.



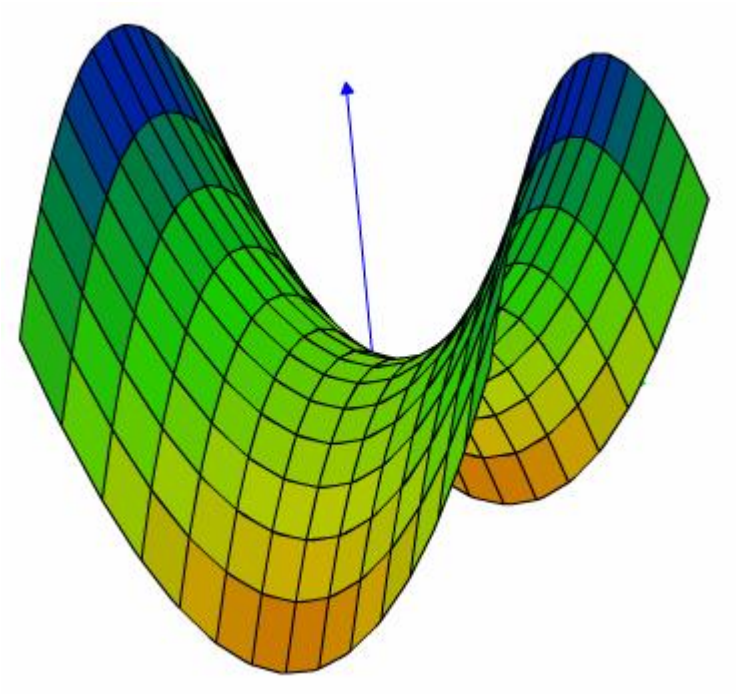
Figur 7.2 Flate fylt med en farge uten rutenett



Figur 7.3 Flate fylt med en farge med rutenett

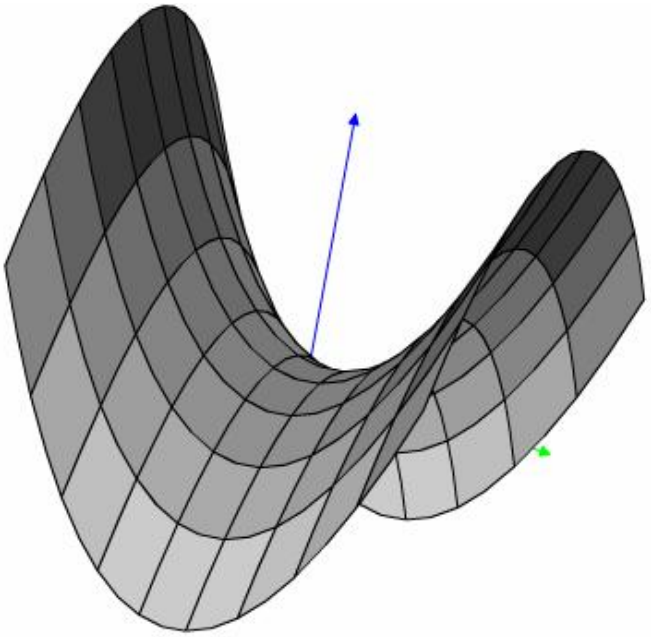
7.2.2 Fylt flate med flere farger

Den vanligste metoden for å representere en farget flate er ved å la hver rute på en rutenett formet flate få en farge basert på egenskapen til denne ruten. Det finnes flere egenskaper som kan benyttes som grunnlag for fargelegging og dette gjør farger til et kraftig verktøy. Eksempler på slike egenskaper kan være høyde, krumming eller retning til normalvektor. Eksempelet på figur 7.4 er fargene blitt definert ut i fra høyden til flaten. En metode som definerer farge med hensyn på høyde kan benyttes på to måter. Den første muligheten er å la applikasjonen beregne hvilke farger som skal benyttes i rutene hver gang flaten blir tegnet opp. Høyden til flate kan enkelt bli regnet ut ved hjelp av minimum og maksimum y verdier til flaten, der hver rute har sin y verdi og dermed en farge. Siden en rute har flere y verdier vil fargen til hver rute bli regnet ut ved å ta midtpunktet mellom minimum og maksimum y verdi til ruten. Dette vil gi de laveste rutene en rød farge mens de høyeste rutene får en blå farge. Denne metoden fungerer fint så lenge flaten kun blir rotert rundt y aksen. Dersom flaten blir rotert rundt z eller x aksen kan, og sannsynligvis vil, avstanden mellom y min og y maks forandre seg. Hvis denne avstanden blir mindre vil fargeendringen per y-verdi forandre seg og gi en skarpere fargeovergang mellom hver rute. Samtidig vil rutene få en annen farge enn den først ble tildelt. Et eksempel på dette er om en flate blir rotert 90 grader rundt z aksen. Før rotasjon ville den øverste ruten ha en sterk blå farge, men etter rotasjon vil ikke y verdien til denne ruten lenger være den høyeste. Dette innebærer at en annen rute har tatt over den høyeste fargen. Flaten vil da bli vanskeligere å tolke, siden ingen rute har en fast farge. Den andre måten å organisere fargelegging ved høyde er å kun bestemme hvilke farger de ulike rutene skal ha en gang. Dette blir gjort når flaten genereres og hver farge blir lagret sammen med koordinatene. Når flaten blir rotert vil det være lettere å tolke hvilken rute som eventuelt lå øverst ved oppstart. Et annet positivt element med denne metoden er hurtighet. Det er ikke nødvendig å kalkulere fargene hver gang det skjer en endring på flaten.



Figur 7.4 Flate representert med flere farger

Det er også mulighet for å stille inn flere typer fargemønster, for å skille egenskaper eller for gi muligheter til å velge et mønster som kan gi bedre forståelse. Stor spredning av farger kan også være misvisende på enkelte flater. En annen form for fargefordeling er å kun benytte forskjellige grå toner til å skille rutene. Grå toner er en av mange mønster som kan benyttes. Andre mønster som kan benyttes for å definere flaten er for eksempel nyanser i grønt, rødt, gult eller blått for å nevne noen.



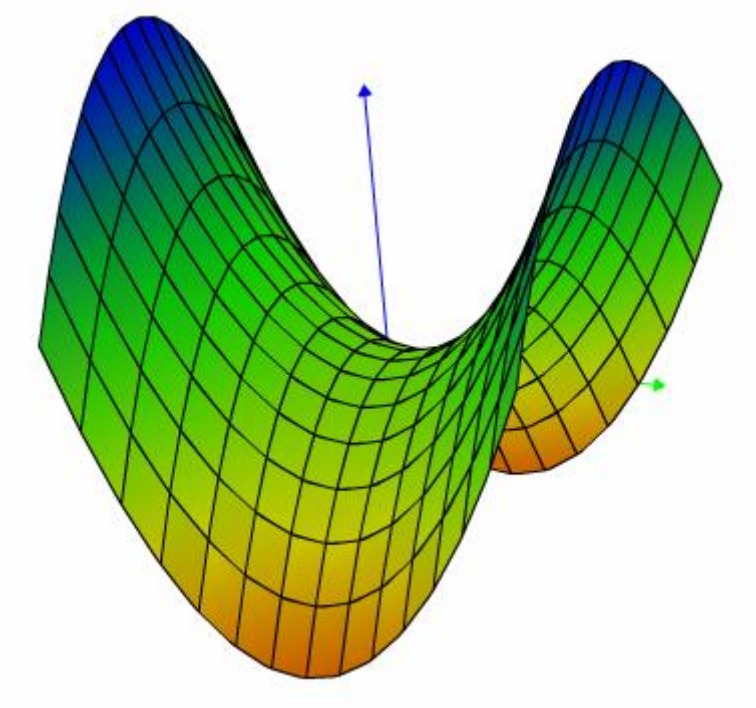
Figur 7.5 Flate representert med gråtoner

Det negative med fyllfarger er at de er tunge å tegne opp. Dette gjelder spesielt på områder som rotasjon, skalering og translasjon der koordinatene blir endret og krever at hele flaten må tegnes opp på nytt. Forsinkelsen avhenger også av hvor detaljert rutenettet til flaten er. Som diskutert i kapittelet om flater ga rotasjon av et veldig detaljert rutenett forsinkelser i opptegningen. Fargelegging forsterker derfor forsinkelsen når grensene til flaten blir større og rutenettet blir detaljert. Fargelegging i Flash ga såpass store forsinkelser at det måtte ses på metoder for å minske denne forsinkelsen. Den viktigste grunnen til at en flate blir rotert er at det er ønskelig å se flaten fra en annen vinkel. Frem til flaten har blitt rotert til en ønsket posisjon har ikke fargene noen stor virkning. Derfor blir kun rutenettet til kurven tegnet opp i tidsrommet når rotering pågår. Etter flaten er rotert i ønsket posisjon blir rutene fylt med farge igjen. Denne metoden viste seg å være svært effektiv, og kom spesielt til syne under rotasjon ved bruk av mus.

7.2.3 Fylt flate med gradient

En flate representert med få ruter, vil ha større krav til fargevalg enn en flate representert med et detaljert rutenett. Anta at en flate består av et antall ruter som går mot uendelig. Hver rute vil da bestå av et lite punkt. Hele flaten skal fylles med farger fra spekteret rødt til blått, ut i fra laveste til høyeste y verdi. Hvert punkt vil da få tildelt en farge ut i fra y koordinaten. Dette vil gi korrekt farge siden det ikke er mulig å definere noe mindre rutestørrelse. Ettersom størrelsen til ruten øker, vil også unøyaktigheten til fargen bli større. Dette vil si at en farget flate som benytter et 100x100 rutenett vil ha mer korrekt fargefordeling enn en flate som er representert med et 10x10 rutenett. En flate som benytter få ruter har allikevel mulighet til å presentere en detaljert fargefordeling ved å bruke gradient.

En gradient er en overgang mellom to eller flere farger. Antall farger som blir benyttet kommer helt an på hvordan fargeovergangen skal være. Dersom en farge skal gå fra rødt til blått og deretter til rødt igjen, krever dette tre fargers gradient. En startfarge, rød, en senterfarge, blå, og en sluttfarge som er rød. Hvis det kun dreier seg om mellom to farger kreves det kun en startfarge og en sluttfarge. En rute som har flere y verdier skal representeres ved flere farger, dersom y verdien styrer fargene. Fargene vil da gå fra laveste y til høyeste y verdi. Ved å finne minimum og maksimum y verdi til en rute er det mulig å legge en gradient mellom disse verdiene. Ruten vil da bestå av flere farger, avhengig av størrelsen og dermed gi en mer korrekt fargefordeling.



Figur 7.6 Flate med gradient

ActionScript tilbyr tre forskjellige metoder for å gradientfylle et område. Alle metodene gir samme resultat, men tar ulike parametere ut i fra hvilken informasjon som er tilgjengelig. Det kreves også en del basis informasjon om fargene som gjelder for alle metodene. Metoden som fargelegger med gradient heter beginGradientFill() og tar fillType, colors, alphas, ratios og matrix som obligatoriske parametere. Beskrivelse av disse er vist i tabell 7.1 nedenfor.

Tabell 7.1 Obligatoriske parametere til beginGradientFill()

Obligatoriske parametere til beginGradientFill() metoden	
Parameter:	**Beskrivelse:**
fillType	Må enten settes til ”linear” eller ”radial” og definerer hvilket mønster/fordeling fargene skal ha. Skal være av type String.
colors	Fargene som skal benyttes i fyllingen. Skal være av type Array.
alphas	Synbarheten til fargene. 0 tilsier ikke synlig og 100 er fult synlig. Må stå i samsvar med antall farger i colors. Skal være av typen Array.
ratios	Fordelingen av fargene. Tar verdier fra 0 til 255 og må være i samsvar med antall farger i colors. Skal være av typen Array.
matrix	En matrise som definerer området gradienten skal fylles over. Denne matrisen kan settes opp på tre forskjellige måter.

Flash krever at det settes opp en egen matrise som inneholder informasjon om området som skal gradientfylles og hvilken retning gradienten skal ha.

matrix.createGradientBox(width, height, rotation, tx, ty)

Dette er en matrise som er avhengig av at matrix er av typen Matrix. Matrix er kun tilgjengelig i Flash versjon 8 og kan derfor ikke benyttes av tidligere Flashspillere. Denne metoden må ta imot parameterne bredde og høyde av arealet som skal fylles. Det er mulig å sende med rotasjonsgraden til gradienten som blir målt i radianer og translasjonsvariabler for x og y verdiene. Halvparten av høyden og bredden blir lagt til translasjonsverdiene for å finne grensen der overgangen mellom fargene er.

matrix = {a:, b:, c:, d:, e:, f:, g:, h:, i:}

Denne metoden definerer en 3x3 matrise av følgende form:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} <=> \begin{bmatrix} w & 0 & 0 \\ 0 & h & 0 \\ x & y & 1 \end{bmatrix}$$

der ”w” og ”h” definerer høyden og ”x” og ”y” definerer grensen for overgangen mellom fargene. Denne matrisen kan enkelt brukes sammen med rotasjon og skaleringsmatriser, men har altså ikke en egen rotasjonsvariabel.

matrix = {matrixType:"box", x:, y:, w:, h:, r:}

Denne metoden definerer fyllområde som en boks. Deretter blir koordinatene (x, y) til det laveste punktet definert. Parameterne ”w” og ”h” definerer bredden og høyden til gradientboksen, mens ”r” definerer rotasjonen av gradienten målt i radianer.

Disse matrisene definerer kun en tenkt gradient flate. Hvilke områder som blir fylt med gradient er avhengig av lineTo() eller curveTo() metodene. beginGradientFill() metoden slutter å fylle når endFill() metoden blir kjørt eller det kommer en moveTo() metode. Dersom ikke lineTo()/curveTo() segmentene danner et lukket område etter endt fylling vil ActionScript selv tegne en linje og lukke området.

Ved implementasjon av gradient farger på et tredimensjonalt objekt er det en utfordring å få rotert gradienten rett vei. Dersom roteringsgraden til disse metodene er null vil retningen på gradienten gå fra venstre mot høyre langs x aksen, der fargen til venstre er første fargen i ”colors” listen. Rotasjon av gradient skjer med klokken. Opp mot eksempelet av høydedefinert fargelegging, må gradienten roteres 90 grader slik at fargene peker oppover langs y aksen. Dersom flaten blir rotert må også gradienten roteres. Hvilket mønster som blir brukt for gradientfylling har også stor innvirkning på om gradienten skal roteres eller ikke. I eksempelet som har blitt benyttet til nå skal ikke gradienten roteres dersom rotasjonen kun går om y aksen. Grunnen til dette er at y verdien styrer fargene og rotasjon rundt y aksen endrer ikke y verdiene til rutene. Siden det ikke er noen endring i y verdiene skal heller ikke gradienten roteres. Gradienten må derimot roteres når flaten roteres rundt x og/eller z aksen, siden denne rotasjonen endrer y verdiene.

7.3 FARGER OG METODER

Det ble diskutert over at en flate fylt med kun en farge kan gi flaten et flatt perspektiv og være med på å skjule ujevnheter i flaten. Den hadde allikevel en positiv effekt ved å gi flaten et helhetlig bilde og skjulte linjene som lå på baksiden av flaten. En flate fylt med flere farger vil også ha disse positive effektene. Forskjellen mellom en ensfarget og flerfarget flate er at en flerfarget flate har mulighet til å fremheve eventuelle ujevnheter på flaten i stedet for å skule dem. Samtidig er en flerfarget flate med på å gi flaten et mer romlig bilde. Dette vil si at bruk av flere farger har de samme positive egenskapene som ved bruk av en farge, samtidig som alle de negative effektene er fjernet og blitt omformet til positive effekter. Selv om bruk av flere farger krever noe mer kalkulasjon er kalkulasjonstiden til fargefordelingsmetoden såpass rask at denne negative siden ikke blir vektet nok. En funksjon som fargelegger en flate med en farge er derfor ikke nødvendig i 3D modulen. Bruk av et flerfarget system representerer flatens egenskaper bedre etter hvor detaljert rutenettet til flaten er. Dersom rutenettet blir veldig detaljert vil flaten få tilnærmet lik fargefordeling som om gradient farger skulle bli benyttet. Problemet med et slikt detaljert rutenett er at ActionScript får problemer med alle kalkuleringene og fargeleggingen. Det vil derfor aldri være mulig å benytte et så detaljert rutenett, dersom flaten ikke har svært enkel oppbygning. Gradientfarger løser dette problemet. Dersom det presenteres en stor og avansert flate, må antall ruter i rutenettet ned til et minimum for å holde effektiviteten og hurtigheten høyst mulig. I dette tilfelle vil det være mer hensiktsmessig å benytte gradient farger siden disse vi gi flaten en glatt og fin overgang mellom hver rute. En annen grunn til å benytte gradient på store ruter er muligheten for å benytte enten lineær eller radial gradientfordeling. Eksempel på bruk av radial er dersom en stor rute inneholder en liten høydeforskjell midt på flaten. Denne høydetoppen kan komme frem ved å benytte radial gradientfarge ut i fra punktet som definerer toppen av høyden. Det er allikevel en del utfordringer rundt bruk av gradient. Dersom en flate blir rotert må også retningen på gradienten bli rotert. Hvor avansert denne utregningen blir, er avhengig av fargefordelingsmetoden som blir benyttet. Roteringsvariabelen må i tillegg testes opp mot retningen(e) flaten blir rotert i. Det er svært viktig at gradienten får korrekt retning. Dersom retningen til gradienten er gal, etter at flaten er blitt rotert vil fargene mislede brukeren i stedet for å gi en pedagogisk effekt. Slik at dersom kalkuleringen av rotasjonsgraden til

gradienten blir for avansert og tung, og i tillegg kan gi feil informasjon i enkelte posisjoner, bør heller vanlig flerfarget system benyttes. I tillegg er det mulig at kalkulasjonstiden for rotasjonsgraden tar like lang tid som om rutenettet skulle være mer detaljert. Det vil da være et spørsmål om det er mer hensiktsmessig å benytte et tettere rutenett og et vanlig flerfarget system enn bruk av gradient. Svaret på dette spørsmålet er avhengig av flaten som blir presentert og hvilke egenskaper som mønsteret representerer. Hvilken type fargemønster som blir brukt har også innvirkning på om det bør benyttes gradient eller ikke. Et eksempel på dette kan være bruk av grå toner til å skille y verdiene til en flate. Siden antall mulige gråtoner skal fordeles til en y verdi innefor maks og min y, vil dette gi mildere fargeovergang mellom fargene enn om fargespekteret skulle gå fra blått til rødt. Dette vil si at fargeovergang mellom to ruter vil bli så mild at fargefordelingen vil gi nesten samme resultat som ved bruk av gradient. Det vil da være mest hensiktsmessig å bruke et flerfarget system, siden dette krever færre kalkulasjoner, men igjen vil antall ruter i rutenettet ha innvirkning. Dersom flaten er avansert og rutenettet består av få ruter, vil gradient være den beste løsningen. Dersom flaten skal ha et detaljert rutenett og en fargefordelingsmetode som det er vanskelig å gi en fargeretning på vil et flerfarget system være den beste løsningen. Begge metodene er derfor implementert i 3D modulen.

Egenskapene til en flate kan representeres ved hjelp av ulike fargemønster, men bør ett fargemønster låses til en egenskap? Gå ut ifra et eksempel der fargen til hver rute på en flate blir definert ut i fra høyden til ruten. Til å presentere denne høyden kan det benyttes ulike mønster. Ved bruk av et fargespekter fra rødt til blått, kan den blå fargen representere den høyeste delen av flaten og rød definere den laveste delen. Men høyden kan også representeres ved et gråtone mønster, der lysest grå er den laveste delen og den mørkeste gråfargen representer den høyeste delen av flaten. Hvilke av disse metodene som egner seg best til å representere høyden til en flate er forskjellig fra person til person. Det bør derfor være mulig å stille inn det mønsteret som brukeren mener gir størst forståelse. Neste egenskap til en flate kan være å se på z verdiene til hver rute. Det kan også i dette tilfellet være ønskelig å benytte et fargespekter fra blått til rødt, der blått definerer positiv z verdi og rødt representerer negativ z verdi. Dersom et mønster skal være låst til en egenskap vil det ikke være mulig å benytte spekteret rødt til blått fordi dette er brukt i høydeegenskapen til flaten. Et fargemønster kan derfor benyttes til forskjellige

egenskaper og det vil være opp til brukeren å definere hvilket som gir størst mening.

7.4 PEDAGOGIKK OG FARGER

Farger er et sterkt virkemiddel som kan brukes til å skille ulike elementer fra hverandre eller binde elementer sammen. Farger har også den egenskapen at de kan være med å utheve egenskaper, som uten farger er vanskelige å legge merke til. Eksempelet med høydebasert fargelegging gjør det enklere å tolke hva som er topp og bunn av flaten etter flaten har blitt rotert i forskjellige posisjoner. Samtidig er den med på å gi flaten et helhetlig bilde der sammensetningen av rutene kommer klarere frem. En annen positiv effekt av farger opp mot rutenett er at den skjuler baksiden av flatene. Dette er med på å gjøre tolkningen av en flate enklere.

8 INTERAKSJON OG OPPSETT

8.1 OPPSETT AV KOORDINATSYSTEM

Hvert 3D objekt som blir projisert har senter i det globale koordinatsystemet, foreløpig finnes det kun ett koordinatsystem i 3D modulen. Som forklart tidligere i oppgaven, vil alle figurer bli tegnet opp i ett movieClip. Dette movieClipet blir forflyttet slik at movieClipets origo vil være midt i opptegningsområdet slik det ser ut for brukeren.

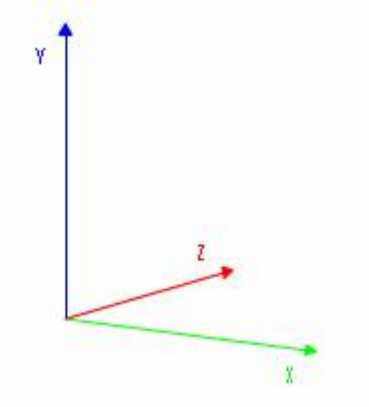
For at brukeren skal se hvordan 3D objektet er tegnet i rommet, tegnes det opp en akse i hver retning, x, y og z, hvor aksenes utgangspunkt er senteret for 3D objektet.

8.1.1 Orientering av aksene

Retningen til de ulike aksene kan være forskjellige, men de må oppfylle høyrehåndsregelen.[34]

Retningen for de ulike aksene i 3D modulen er valgt med tanke på hvordan brukeren er vant med å se ett koordinatsystem. Brukeren vil mest sannsynlig kjenne best til 2D koordinatsystemer, hvor y aksen har positiv retning oppover og x aksen positiv retning mot høyre. Derfor har disse retningene også blitt valgt når 3D objektet tegnes opp første gangen. I tillegg til at den tredje aksen, z aksen, vil ha positiv retning innover i skjermen grunnet høyrehåndsregelen.

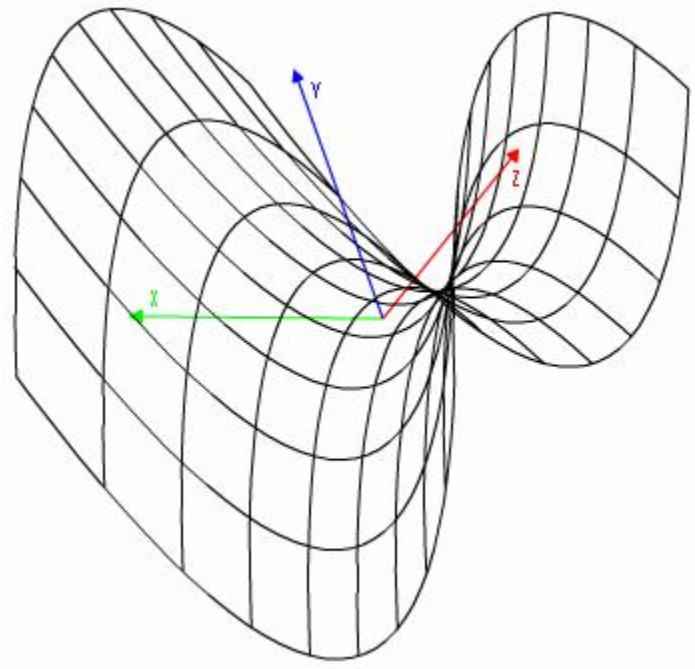
Aksene vil forandre retning ettersom 3D objektet roteres i ulike retning. Figur 8.1 viser opptegnede akser.



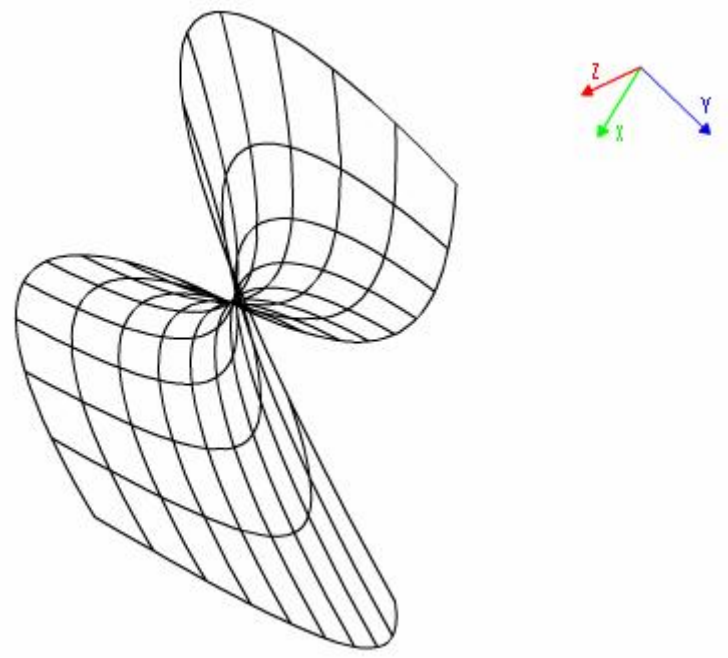
Figur 8.1 Akser i rommet

8.1.2 Posisjon av aksene

Innledningsvis ble det sagt at utgangspunktet for aksene er i senteret til 3D objektet, for noen situasjoner vil dette være gunstig mens for andre ikke så hensiktsmessig. Til nå i utviklingen av 3D modulen tegnes aksene alltid opp med utgangspunkt i 3D objektets senter, vist under på figur 8.2. Ett annet alternativ vil være å tegne opp aksene utenfor eller ved siden av 3D objektet, på en slik måte at aksene ikke interfererer med 3D objektet. Se figur 8.3.



Figur 8.2 Isometrisk projeksjon hvor aksene er tegnet sammen med objektet



Figur 8.3 Perspektivisk projeksjon hvor aksene er tegnet ved siden av objektet

Det vil være avhengig av hensikten med 3D objektet i undervisningssammenheng, som bestemmer hvilke av de to plasseringene som passer best.

8.1.3 Skala på aksene

Den virkelige størrelsen på et 3D objekt er det tilnærmet umulig å se ut i fra en projeksjon. Defineres det en skala på aksene, som kan vise hvor i koordinatsystemet 3D objektet befinner seg, er det mulig å trekke slutninger på hvor stort eller lite et 3D objektet vil være i virkeligheten når målestokken for skalaen er gitt. Men, som for posisjonering av aksene, er det ikke nødvendig med en skala for alle situasjoner.

Trekkes aksene ut fra 3D objektet og plasseres ved siden av er det kanskje lite hensiktsmessig å benytte skala. I hvert fall med tanke på å bruke skalaen til å finne hvor en gitt del av 3D objektet befinner seg i koordinatsystemet.

8.2 TRANSLASJON AV 3D OBJEKTER

Applikasjonen skal gi brukeren mulighet til å flytte et 3D objekt innenfor visningsområdet. Dersom aksene ikke blir tegnet opp ved siden av 3D objektet, følger disse med.

8.3 SKALERING AV 3D OBJEKTER

Skalering av 3D objekter kan være nødvendig dersom det ble projisert i ett lite område av visningsområdet eller brukeren ønsker å se nærmere på detaljer i 3D objektet.

Skaleres aksene sammen med objektet vil det bli en forstørrelse eller forminskelse av objektet. Det vil si at ett punkt i 3D objektet vil ha samme verdi etter skaleringen som den hadde i utgangspunktet.

Blir 3D objektet skalert og ikke aksene, vil hvert punkt som blir skalert endre posisjon i forhold til det globale koordinatsystemet. Punktene vil allikevel ikke endre posisjon i forhold til formen av flaten.

8.4 ROTASJON AV 3D OBJEKTER

Det finnes forskjellige muligheter for å rotere et objekt i en 3D modul. De mest vanlige og relevante mulighetene vil bli gjennomgått nedenfor.

8.4.1 Rotasjon rundt et fast aksesystem

Denne metoden roterer objektet rundt et aksesystem som ikke forandrer seg. Man kan da si at aksesystemet er låst eller fast, og det er bare objektet som blir rotert.

Et eksempel på metoden kan være at man har et objekt som blir rotert 45 grader rundt x aksen. Da vil objektet ha en rotasjon på 45 grader i x retning, mens koordinatsystem ikke er forandret. Objektet skal deretter roteres 180 grader rundt y aksen. Objektet vil ha samme x verdier som før rotasjonen, men vil nå være negative.

8.4.2 Rotasjon rundt et aksesystem som følger rotasjonen

Denne metoden går ut på at både koordinatsystemet og aksene i modulen roteres. Det vil si at metoden gjør det mulig å holde seg til den opprinnelige ligningen som objektet er representert ved, i forhold til koordinatsystemet.

Et eksempel på metoden kan være at man har et objekt som blir rotert 45 grader rundt x aksen. Da vil hele koordinatsystemet være rotert 45 grader rundt x aksen, og y aksen vil da få en helning innover i skjermen. Når da objektet skal roteres 180 grader rundt y aksen, vil objektets punkter ha sammen x, y og z verdier som før rotasjonen i y retning.

Resultatet ville da ikke blitt det samme som hvis disse opperasjonene hadde blitt gjennomført med et fast aksesystem.

8.4.3 Rotasjon rundt lokale akser

Denne metoden går ut på å definere et tenkt lokalt koordinatsystem til hvert objekt. Anta at dette koordinatsystemet er fast i forhold til objektet. Dette vil si at koordinatsystemet følger med objektet dersom objektet skulle bli flyttet i forhold til det globale koordinatsystemet. Disse aksene er allikevel faste i forhold til at de ikke endrer retning. Objektet kan nå roteres rundt sitt lokale koordinatsystem.

Anta at en linje med lengde lik 10, er visualisert langs x aksen med midtpunkt i posisjon x = 100, y = 50 og z = 0 i det globale koordinatsystemet. Anta videre at denne linjen har et tenkt lokalt koordinatsystem der origo er midtpunktet til linjen. La det skje en lokal rotasjon av linjen på 90 grader i z retning. Etter rotasjon vil midtpunktet til denne linjen fortsatt ha midtpunkt i (100, 50, 0), men retningen til linjen vil nå være langs y aksen i stedet for x aksen. På denne måten kan et objekt bli rotert i forhold til sitt eget koordinatsystem uten at det globale koordinatsystemet har innvirkning på rotasjonen.

8.5 INTERAKSJON

3D objektet som visualiseres i 3D modulen vil ha liten nytte i undervisnings sammenheng dersom ikke brukeren kan gjøre noe med det. De mulighetene 3D modulen gir brukeren for å manipulere 3D objektet er foreløpig translasjon, rotering og skalering. Hvordan brukeren kan interaktere for å utføre disse manipulasjonene er mange. Videre i rapporten nevnes scrollbarer, mus og tekstbokser. En annen interaksjons mulighet kan være touch screen hvor brukeren benytter fingrene på skjermen for å utføre manipulasjonene.

8.5.1 Scrollbarer

Ved bruk av scrollbarer kan brukeren rotere rundt alle de tre aksene. Musa benyttes på scrollbarene for å rotere rundt en eller flere akser. Brukeren vil få god kontroll over roteringen og kan rotere 3D objektet tilnærmet slik han ønsker i alle retninger.

Scrollbarer egner seg bra for å sette begrensninger for hvor mye brukeren kan rotere rundt en gitt akse. Dersom en ikke ønsker å sette begrensninger for brukeren vil det være lite hensiktsmessig å benytte scrollbarer, dette fordi en scrollbar ikke kan være uendelig lang.

Scrollbarer kan også benyttes til forflytning av 3D objektet, men de vil ha begrensninger på lik linje for bruk ved rotasjon. En scrollbar for hver akse kreves, hvor hver av de beveger 3D objektet langs sin respektive akse.

Skalering krever kun en scrollbar, hvor midten av scrollbaren representerer den opprinnelige størrelsen på 3D objektet.

8.5.2 Mus

Musa setter begrensninger for hvor mange akser en kan rotere rundt siden den kan kun beveges i 2D planet. Det er mulig å benytte ett musehjul, dersom en har det, for å rotere rundt den tredje aksen.

3D modulen gir denne muligheten til brukerne, ved at brukeren holder inne venstre museknapp mens han/hun beveger musa for å rotere. Applikasjonen

benytter ikke musehjulet til rotering. I det brukeren presser inn venstre museknapp tar applikasjonen vare på det punktet musa var i da. Beveges musa en gitt avstand fra dette punktet, roteres 3D objektet rundt en eller flere akser bestemt av hvilken vei musa ble flyttet. Så lenge musa ikke befinner seg i nærheten av punktet der hvor venstre museknapp ble presset, vil 3D objektet fortsette å rotere. Det kreves litt trening for å få dette til og det kan være vanskelig å få rotert 3D objektet akkurat slik en ønsker.

Forflytning av 3D objektet med mus vil ha den samme problematikk som ved rotering, musa beveger seg i 2D planet. Løsningen vil være å bruke en kombinasjon av tastatur og musa.

Skalering ved bruk av mus skjer ved hjelp av mushjulet.

8.5.3 Tekstbokser

Tekstbokser for hver akse vil være den metoden som gir brukeren mest frihet for rotering. Brukeren kan selv taste inn grader eller radianer for hvor mye han/hun ønsker å rotere rundt de gitte aksene.

Både for forflytning og skalering er det ikke noe problem å benytte tekstbokser. Det er den metoden som gir brukeren mest kontroll over resultatet.

8.5.4 Interaksjon opp mot parametriserte objekter

Parametriserte funksjoner har den egenskapen at den benytter parametere i stedet for uavhengige variabler. Det blir derfor definert egne funksjoner for x, y og z. Dette gjør det enklere å representere ulike objekter. Det vil være mulig å stille inn ulike verdier for disse parameterne og dermed endre formen til objektet.

8.6 OPTIMALISERING AV FLATER VED ROTERING, SKALERING OG TRANSLASJON OVER TID

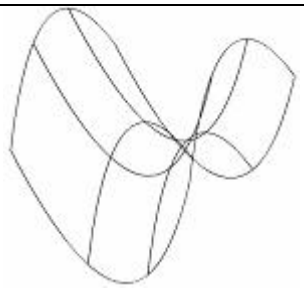
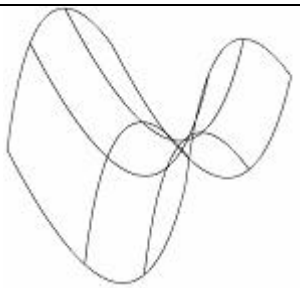
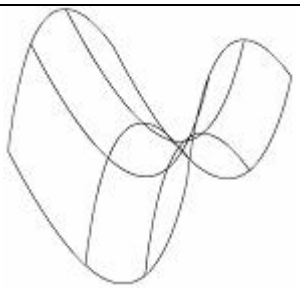
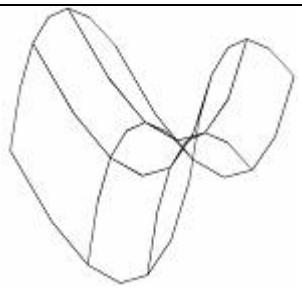
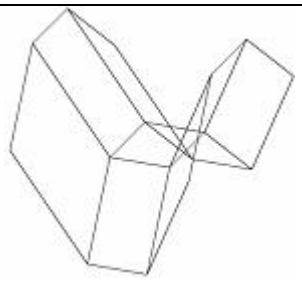
Flatene som blir generert blir definert ut fra et gitt antall punkter. Mellom disse punktene blir det tegnet opp linjer eller kurver avhengig av om lineTo() eller curveTo() blir benyttet. I utgangspunktet er avstanden mellom punktene 1. En så kort avstand vil gi den mest nøyaktige fremstillingen av objektet og være nødvendig ved matematiske beregninger på objektet, men flaten vil bestå av ett stort antall punkter. Er det et vesentlig antall punkter vil det medføre en forsinkelse ved opptegning av skalerte, roterte og transelerte objekter. Er det for stor forsinkelse vil roteringen, skaleringen eller translasjonen virke ujevn eller hakkete, når dette skjer over tid. For en flate med få ruter vil nødvendigvis ikke den hakkete følelsen merkes, men etter hvert som antall ruter øker vil det bli mer merkbart. Dette kommer av at antall punkter øker betraktelig etter hvert som antall ruter øker.

3D modulen vil ha klare begrensninger med tanke på visualisering av objekter, dersom det hele tiden må være et minimum antall ruter. Årsak er at mange objekter krever en detaljrik fremstilling, hvor det er nødvendig med flere ruter. For å motvirke den ujevne eller hakkete følelsen, er det nødvendig med en optimalisering av hvordan flatene blir beregnet og tegnet opp.

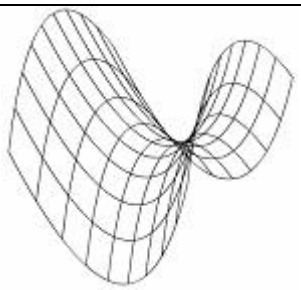
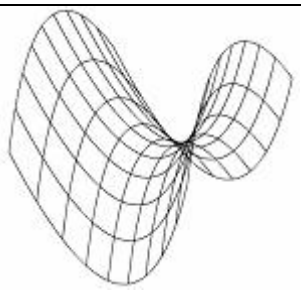
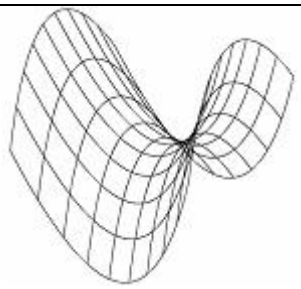
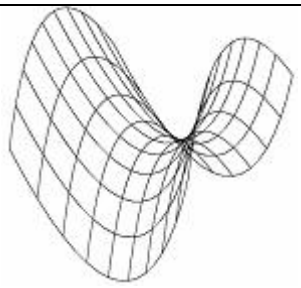
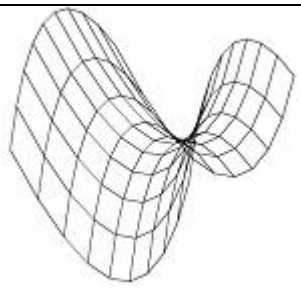
Tidligere i dette delkapittelet ble det sagt at avstanden mellom punktene var 1, ved å øke denne avstanden x antall ganger vil antall punkter reduseres betraktelig. Denne optimaliseringen vil gjøre for eksempel roteringen jevnere, men på bekostning av nøyaktigheten til den representerte flaten.

Det har blitt foretatt testing på den hyperbolske parabelen, hvor antall horisontale og vertikale linjer og avstanden mellom punktene varieres. 4, 10 og 20 horisontale- og vertikale linjer er bitt undersøkt. Avstanden mellom punktene vil påvirke antall linjesegmenter som blir tegnet opp for hver side i rutene. De to siste radene i tabellene viser henholdsvis 1 og 2 linjesegmenter. Under testen ble objektet rotert over tid for å definere hvilke resultater som gav tilfredsstillende jevn rotering, dette kommer frem under kolonnen: Rotering over tid. I tillegg viser kvalitet - kolonnen om opptegningen av objektet var tilfredsstillende i forhold til subjektive meninger. Tid representerer tiden 3D modulen brukte for å prosessere rotering. Testen ble utført på en laptop med en Intel Pentium M prosessor 1600MHz og 1 GB RAM. Tabellene under, 8.1, 8.2 og 8.3, viser resultatene fra testingen.

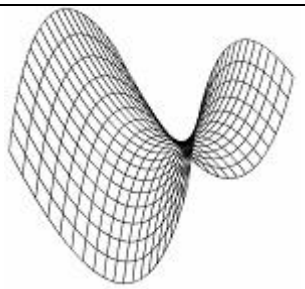
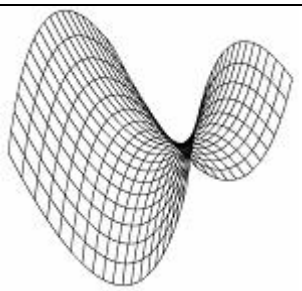
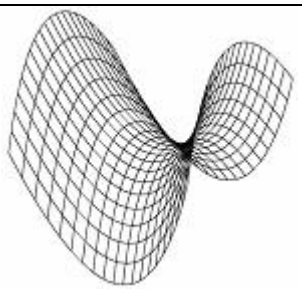
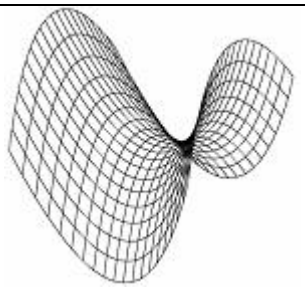
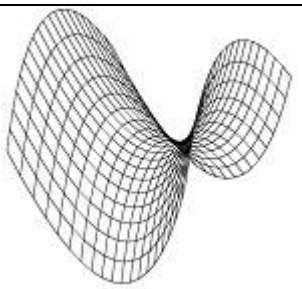
Tabell 8.1 4 horisontale- og vertikale linjer

Rotering over tid	Kvalitet	Avstand	Antall punkter	Tid (ms)	Opptegnet objekt
Nei	Ja	1	3168	617	
Nei	Ja	2	1620	451	
Ja	Ja	8	432	131	
Ja	Nei	44	108	43	
Ja	Nei	87	72	30	

Tabell 8.2 10 horisontale- og vertikale linjer

Rotering over tid	Kvalitet	Avstand	Antall punkter	Tid (ms)	Opptegnet objekt
Nei	Ja	1	9720	1650	
Nei	Ja	2	5184	915	
Nei	Ja	8	1620	341	
Ja	Ja	15	972	266	
Ja	Nei	29	648	178	

Tabell 8.3 20 horisontale- og vertikale linjer

Rotering over tid	Kvalitet	Avstand	Antall punkter	Tid (ms)	Opptegnet objekt
Nei	Ja	1	21660	4043	
Nei	Ja	2	11552	2034	
Nei	Ja	4	7220	1271	
Nei	Ja	7	4332	785	
Nei	Ja	14	2888	559	

Ut i fra resultatene fra testen, representert i tabellene på de forrige sidene, vil en flate visualisert med 10 x 10 linjer og 2 linjesegment for hver side i rutene være den som visualiserer flaten mest nøyaktig hvor roteringen er tilfredsstillende. I samme rad finner en tiden prosesseringen brukte, 266ms. Det vil si at alle transformasjoner som skal skje over tid, må ha en raskere eller lik prosesserings tid enn 266ms for at det ikke skal virke ujevnt eller hakkete, på en maskin tilsvarende testmaskinen.

8.7 NORMALVEKTOR

En normalvektor er en vektor som står 90º (normalt) på noe annet, eksempelvis en annen vektor, en linje eller et plan. [35]

Dersom flaten er krummet vil normalvektoren til et punkt p stå normalt på punktet i forhold til tangentplanet til det 3-dimensjonale punktet.

8.7.1 Beregning av normalvektor

Normalvektoren til et punkt p regnes ut ved å krysse to vektorer som ligger på tangentplanet til punktet. Normalvektoren kan ha retning ut fra punktet både i positiv eller negativ retning. Hvilken retning som skal vises må velges ut fra orienteringen av koordinatsystemet til flaten.

normalvektor.x = vektor1.y * vektor2.z - vektor2.y * vektor1.z
normalvektor.y = vektor2.x * vektor1.z - vektor1.x * vektor2.z
normalvektor.z = vektor1.x * vektor2.y - vektor2.x * vektor1.y

Deretter må vektoren normaliseres. Dette gjøres ved først å regne ut lengden av normalvektoren. Deretter blir denne lengden delt på hver x, y og z koordinat.

normVektorLengde = $\sqrt{}$((normalvektor.x)² + (normalvektor.y)² + (normavektor.z)²)

Den ferdig normaliserte vektoren blir da:

Vektor.x = normalvektor.x / normVektorLengde
Vektor.y = normalvektor.y / normVektorLengde
Vektor.z = normalvektor.z / normVektorLengde

8.7.2 Valg av kalkuleringspunkt

Normalvektoren må regnes ut i fra et punkt på en tredimensjonal figur. Brukeren bør selv kunne velge hvor normalvektoren skal vises og dette valget kan gjennomføres på to måter. Enten så kan brukeren velge et gitt tredimensjonalt punkt som eksisterer på flaten eller benytte rutenettet til flaten. Ved å benytte rutenettet kan brukeren velge ut en rute som det er ønskelig å få visualisert normalvektoren i. Ruten vil da selv velge punktet som skal kalkuleres.

Uavhengig av hvilken metode som benyttes over, må brukeren interaktere opp mot flaten. Et eksempel på interaksjonsmetode er bruk av tastatur. En bruker vil da kunne skrive inn en x, y og z koordinat som eksisterer på flaten. Dette punktet vil da enten representere selve kalkuleringspunktet eller en rute. Rutenettet tilbyr også muligheten for å velge en spesifikk rute ved hjelp av kolonne og rad nummer til ruten. Den vanligste interaksjonsmetoden opp mot et grafisk grensesnitt er å benytte en mus. Denne gir mulighet til å velge et punkt eller en rute på den visualiserte flaten.

8.8 DISKUSJON

8.8.1 Koordinatsystem

Aksene i koordinatsystemet skal hjelpe brukeren i å orientere 3D objektet i rommet, ved hjelp av retningen til aksene og skalaen som vises på aksene.

Den generelle bruker som 3D modulen er utviklet for har mest sannsynlig jobbet lite med visualisering av objekter i matematiske sammenhenger. Da er det viktig å bygge videre på hva brukeren kjenner til fra tidligere undervisning, nærmere bestemt koordinatsystemer i 2D planet. Her representeres x aksens positiv retning

mot høyre og y aksen oppover. Ved å benytte disse retningene for x og y aksene i 3D planet må z aksen ha positiv retning innover i skjermen. For å skille aksene mer fra hverandre enn bare ved å navngi de med x, y og z, vil hver akse få en egen farge. X aksen vil være grønn, y aksen blå og z aksen vil få en rødfarge.

Roteres objektet vil aksene vise hvordan objektet er orientert i rommet. I denne sammenheng vil det være mer oversiktlig å tegne aksene utenfor eller ved siden av objektet. Dette fordi aksene kan bli ”gjemt” i objektet, når det tegnes opp med farger.

Ved matematiske beregninger og beregninger ut i fra det isometriske projiserte objektet kan det være hensiktsmessig å ha skalaer på aksene som gjør det mulig å avlese hvor ett punkt eller en del av objektet befinner seg i rommet. I denne sammenhengen er det nødvendig at aksene er tegnet opp sammen med objektet.

3D modulen vil benytte en kombinasjon av hvor aksene tegnes opp. Det vil hele tiden være opptegnet akser ved siden av eller utenfor objektet, men disse vil ikke ha påtegnede skalaer, og brukeren vil få mulighet til å vise eller ikke vise akser sammen med 3D objektet, disse aksene har opptegnede skalaer.

8.8.2 Rotasjon

Det finnes mange måter å rotere på. To av disse metodene er å rotere objekter der aksene er faste og der aksene følger med rotasjonen. Det finnes både negative og positive sider med begge disse alternativene. Et viktig element i rotasjon, der aksene følger rotasjonen, er at originalkoordinatene til objektet ikke blir forandret. Dette fører til at ligningen vil være den samme under rotasjonen. Hvis man ser på denne metoden fra en negativ side vil det være vanskeligere å rotere i kun en akse ved bruk av mus, under visse forhold. Det vil også være vanskelig å se hvordan rotasjonen vil bli i visse posisjoner. Disse problemene vil ikke oppstå før objektet har blitt rotert flere ganger. I rotering med faste akser vil ligningen eller utrykket til objektet forandre seg og det er ikke lenger muligheter til å benytte seg av denne. Derimot en av de positive sidene er at det er lettere å bruke mus til rotasjonen, og lettere å forstå hvilken vei rotasjonen går, ut fra posisjonen til objektet. Det er også enkelt å rotere kun rundt en gitt akse. Basert på denne kunnskapen vil den beste løsningen i utviklingen av 3D modulen være å satse på rotasjon av objektet med

aksene låst.

Slik som 3D modulen er bygd opp roteres objektene rundt et aksesystem med utgangspunkt i origo. En annen mulighet hadde vært å la objektet rotere rundt sitt eget aksesystem, i tillegg til et globalt koordinatsystem. Dette vil gjøre det mulig å rotere et objekt rundt sitt lokale koordinatsystem eller alle objektene rundt det globale koordinatsystemet. Derfor er den beste løsningen for 3D modulen å bruke et oppsett der objektet kan roteres rundt sine egne akser, i tillegg til at det vil være mulig å rotere rundt det globale koordinatsystemet.

8.8.3 Interaksjon

Mus, tastatur og scrollbarer er de interaksjonsmulighetene 3D modulen vil gi brukeren. Interaksjon ved hjelp av musa gjør det enklere og mer ”artig” for brukeren å navigere seg rundt i 3D modulen. Operasjoner på 3D objektet vil kunne utføres raskere, da rammene for operasjonene allerede er definert i applikasjonen. Ulempene er blant annet at interaksjon med mus vil gi brukeren begrensninger og optimalisering av posisjonen til ett rotert 3D objekt vil være vanskelig. Dette kan oppleves negativt for de med et høyere kunnskapsnivå til emnet. For de som akkurat har fått kjennskap til emnet vil det være hensiktsmessig å gi brukeren færre valg slik at de grunnleggende egenskapene til visualisering av 3D objekter kommer frem. Det være seg hvordan 3D objektet forandrer seg i rommet avhengig av hvilke operasjoner som utføres.

Scrollbarer vil gjøre det enklere for brukeren å tilpasse operasjoner, som rotering, skalering og translasjon, i de tre ulike retningene x, y og z enn hva som er mulig med mus.

Bruk av tekstbokser vil gi brukeren flest valgmuligheter i tillegg til at operasjonene som blir utført gir ett tilnærmet likt resultat som ønsket i utgangspunktet.

Hvilke interaksjonsmuligheter applikasjon skal tilby brukeren vil være en kombinasjon av alle tre. På den måten vil brukere med ulikt kunnskapsnivå få tilpasset 3D modulen til sitt bruk. Dette løses ved at ulike grensesnitt defineres med tanke på kunnskapsnivå og brukeren kan selv velge hvilket han/hun vil benytte.

8.8.4 Interaksjon opp mot parametriserte objekter

Et parametrisert objekt har som definert tidligere egne funksjoner for x, y og z. Det vil derfor være mulig å stille på de ulike parameterne til funksjonene før objektet blir generert. Etter det parametriserte objektet har blitt generert vil objektet allikevel bli lagret ved hjelp av koordinatene som representerer objektet. Dette fører til at interaksjonsmulighetene som gjelder for objekter generert ut fra vanlige funksjonsuttrykk også vil gjelde for parametriserte funksjonsuttrykk. Dersom parameterne til objektet skal endres må objektet tegnes opp på nytt i utgangsposisjonen og generere nye koordinater. Dette vil si at dersom objektet hadde blitt rotert før endringen av parameterne ble gjort, vil objektet miste posisjonen det hadde blitt rotert til.

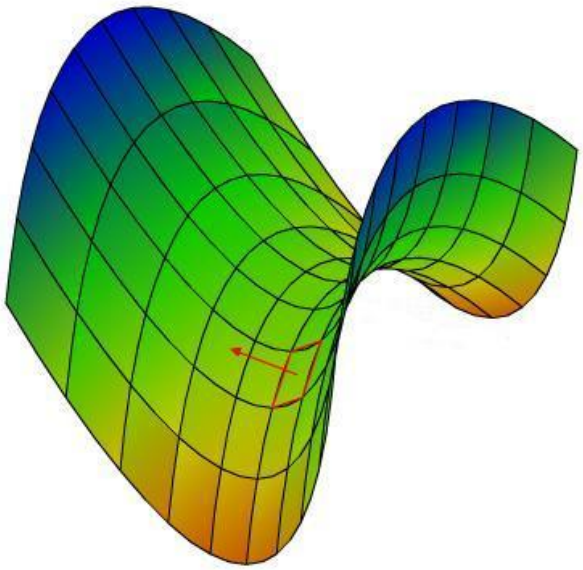
8.8.5 Optimalisering av flater ved rotering, skalering og translasjon over tid

Resultatet fra testen i delkapittelet Optimalisering av flater ved rotering, skalering og translasjon over tid viser at det er nødvendig å redusere kvaliteten til flaten for å få en tilfredsstillende transformasjon. Dette vil si at samme flate som transformeres over tid ikke kan benyttes til matematiske beregninger for å få frem ulike egenskaper til flaten. Alternativet vil være å benytte tekstbokser, som nevnt i delkapittelet Interaksjon, når det skal utføres matematiske beregninger direkte på flaten.

8.8.6 Normalvektor

Som diskutert over finnes det to metoder for valg av kalkuleringspunkt for normalvektor. Den første går ut på å la brukeren velge et punkt fra flaten som det er ønskelig å få kalkulert normalen til. Det positive med denne måten er at brukeren har full kontroll til å velge det punktet som er av størst interesse. Det negative med denne metoden er at det vil være vanskelig å følge en normalvektor dersom den skal bevege seg langs en gitt kurve. Anta det er ønskelig å følge endringen av normalvektoren langs en av linjene i rutenettet til flaten. To muligheter for å velge punktene på flaten vil være å benytte musen eller skrive inn koordinatene ved hjelp

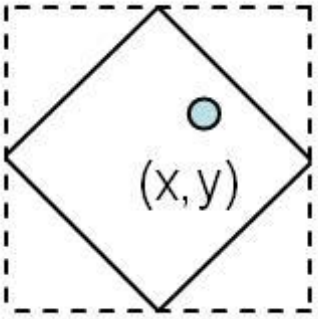
av et tastatur. Muligheten med å benytte et tastatur er ikke en god løsning da det er vanskelig å vite hvilke koordinater som følger linjen, spesielt dersom flaten er blitt rotert. Bruk av mus er et bedre alternativ, men også her er det vanskelig å treffe nøyaktig det neste punktet langs linjen. Det andre alternativet er å benytte rutenettet til flaten som utgangspunkt for hvilket punkt som skal vise normalvektoren. Problemet her er at en rute har et større areal enn et punkt og derfor kan ikke normalvektoren som vises være representativ for alle punktene i ruten. For å finne den gjennomsnittlige normalvektoren for en slik rute, ble de to diagonalvektorene krysset. Ettersom antall ruter i et rutenett går mot uendelig vil normalvektoren bli mer og mer nøyaktig. Et annet aspekt ved å benytte et rutenett er at det er enkelt for brukeren å velge den ruten som normalvektoren skal vises i, ved hjelp av musen. Dette fungerer også effektivt dersom brukeren ønsker å følge en gitt linje. Siden diagonalene til hver rute blir beregnet vil også normalvektoren følge linjen som går igjennom midten av hver rute. Se figur 8.4 på neste side.



Figur 8.4 Normalvektoren til en rute

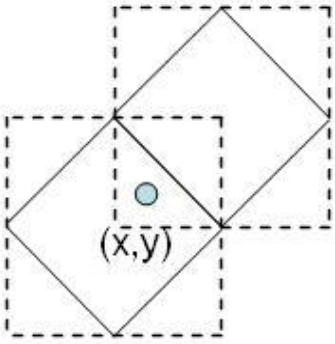
Valg av en rute ved hjelp av mus hadde også noen utfordringer. En mus kan bare sende informasjon til applikasjon om hvilken x og y verdi som har blitt valgt. Siden flaten er 3-dimensjonal og består av x, y og z koordinater kan den valgte x og y verdien gitt av musen representere både fremsiden og baksiden av flaten. Det er ikke

ønskelig å vise normalvektoren for en flate som ikke er synlig og det må derfor testes på z verdien for å konstantere hvilken rute som ligger fremst, dersom flere ruten kan representeres ved valgt x og y verdi. For finne hvilken rute som inneholder valgt x og y verdi, blir det testet på punktene til alle rutene på flaten. Siden hjørnene til hver rute ble lagret i koordinatlisten (beskrevet i lagringskapittelet) er det enkelt å teste på om x og y verdien befinner seg innefor disse hjørnene. Disse hjørnene danner en 2-dimensjonal firkant som vist på figur 8.5.



Figur 8.5 Maks- og min-grensene til en rute

Dette gir en ny utfordring dersom ruten ikke har akkurat samme form som firkanten til ruten. Denne metoden gjør at applikasjonen tror at et xy punkt kan ligge i to ruter som ligger ved siden av hverandre som vist på figur 8.6. En løsning på dette problemet er å la avstanden til midten av en rute bestemme hvilken rute punktet tilhører. Den ruten med minst lengde til senter er punktets eier.



Figur 8.6 Felles område for to ruter

9 RESULTATER

Isometrisk projeksjon egner seg best for 3D modulen. Denne projeksjonen vil gi brukeren mulighet til å avlese posisjonen til et punkt til objektet i rommet, grunnet parallelle linjer som ikke er parallelle i forhold til projeksjonsplanet vil ha lik forkortning.

ActionScript tilbyr de to metodene lineTo() og curveTo() for å tegne linjer og kurver. curveTo() har samme egenskaper som en quadratic Bezier og dette gjør den lite egnet til generering av andre kurvetyper. lineTo() tegner derimot et rett linjesegment og kan derfor benyttes til alle kurvetyper. Det finnes flere metoder for å generere cubic Bezier kurver på. En av disse metodene er midtpunktsalgoritmen som benytter fire quadratic Bezier kurver (curveTo()) til å generere en cubic Bezier kurve. Bruk av curveTo() gjør det vanskelig å følge punktene til kurven og dette fører til at enkelte matematiske metoder kan være vanskelig å regne ut. I tillegg vil denne midtpunktsalgoritmen ikke klare å simulere en cubic Bezier helt nøyaktig. Det vil derfor være enklest å generere en cubic Bezier ut i fra den matematiske formelen til kurven. Cubic Bezier kurver har to kontrollpunkter og gjør det derfor enklere å danne kompliserte egendefinerte flater enn ved bruk av quadratic Bezier kurver.

En flate kan bli tegnet opp på to måter. Enten ved å tegne de horisontale linjene til flaten først og deretter de vertikale eller ved å tegne opp rute for rute. Metoden for å tegne opp de vertikale og horisontale linjene var en rask metode, men tar ikke vare på informasjon og rutene som blir dannet av linjene. Rute for rute metoden tar lengre tid og tegner opp flere linjer to ganger. Den tar allikevel vare på informasjon om hver enkelt rute og er nødvendig dersom flaten skal fargelegges. Dersom rutenettet ble detaljert og hver rute ble tegnet opp ved hjelp av mange linjesegmenter, fikk ActionScript problemer med antall kalkulasjoner.

Singleton Design Pattern løste problemet med flere inkonsistente lagringslister når det eksisterte flere opptegnede objekter i applikasjonen. Alle de opptegnede objektene ble lagret i en array for å gi hurtig tilgang til objektene. På grunn av en allerede presset opptegningshastighet bør eventuelle eksterne punktsett lastes inn i en slik array for å unngå ytterlige forsinkelser.

Quaternion viste seg å være mer effektiv for rotering av objekter i rommet enn matriser. Årsaken til at quaternion er raskere, er at det trengs færre kalkulasjoner for utregningen av de roterte punktene.

Bruk av rutenett til å representere en flate, gir flaten et romlig bilde. En flate som er fylt med en enkel farge gir flaten et ”flatt” perspektiv. Dette kommer spesielt til syne dersom flaten ikke har et synlig rutenett. Rutene i et rutenett kan fylles med ulike farger. Denne metoden er i motsetning til bruk av en fyllfarge med på å fremheve et romlig bilde av flaten. Bruk av flere farger vil derfor være en bedre løsning for visualisering av flater. Gradient har den egenskapen at de kan gi jevne fargeoverganger mellom rutene i et rutenett. Dette innebærer at flaten kan bli representert på god måte også ved bruk av færre ruter. Bruk av flere farger gjør at egenskapene til flaten kan representeres ved hjelp av ulike fargemønster. Et fargemønster blir definert av et begrenset fargespekter.

En kombinasjon av mus, tekstbokser og scrollbarer vil være den beste måten å interaktere opp mot objekter på. Musa vil ha en begrensning i forhold til at den kun registrerer posisjoner i 2D planet og dette gir grunnlaget for kombinasjonen med tekstbokser og scrollbarer. Interaksjon opp mot parametriserte objekter har samme interaksjonsmuligheter som normale objekter. Dette kommer av at det er punktene til flaten som blir lagret og ikke uttrykkene.

Koordinatsystemet har blitt definert med positiv y akse oppover, positiv x akse mot høyre og positiv z akse innover i skjermen. Dette vil gi den beste overgangen fra et todimensjonalt koordinatsystem.

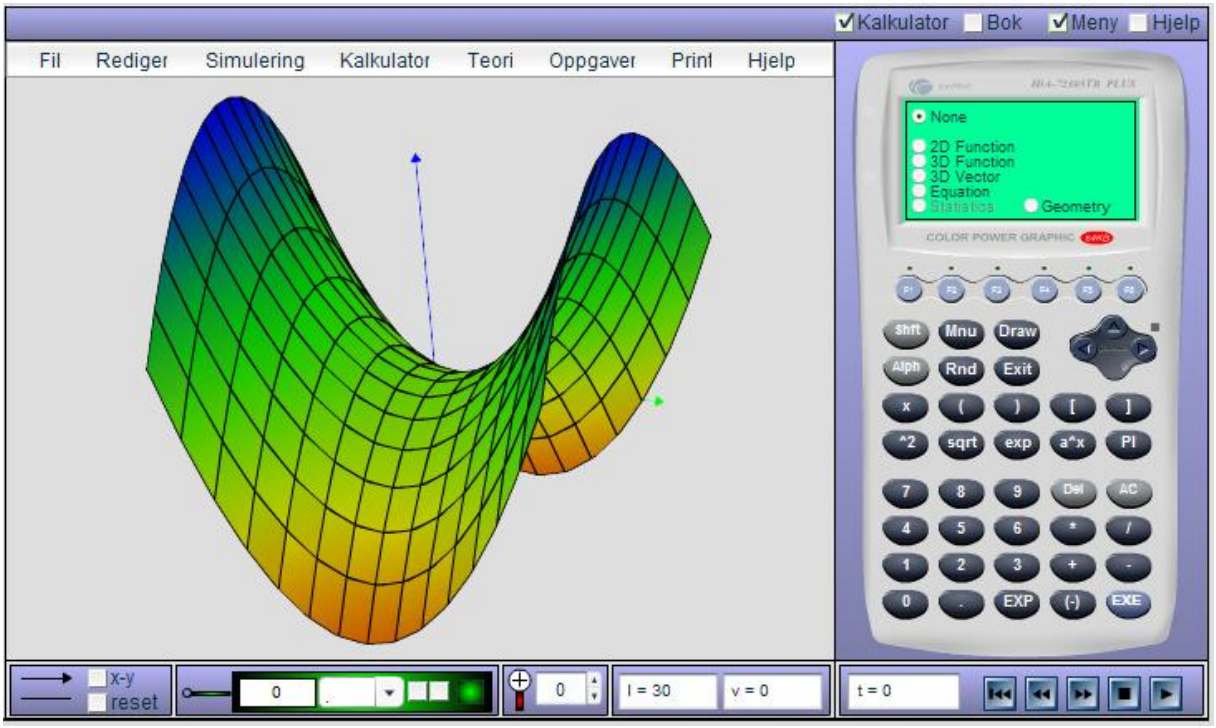
Hvert objekt bør ha sitt eget lokale koordinatsystem, som er definert i det globale koordinatsystemet. Dette vil gjøre det mulig å rotere objektet både i det lokale- og globale koordinatsystemet.

Rotasjon over tid av detaljrike objekter ga stor forsinkelse i prosessering og opptegning og derfor er det nødvendig å optimalisere antall punkter som representerer ett objekt.

Rutenettet er velegnet som utgangspunkt for matematiske beregninger som skal gjøres på flaten. Eksempel på dette er normalvektoren.

Prototypen for 3D modulen har blitt utviklet og implementert i parAbels syntetiske kalkulator, se figur 9.1. Prototypen kan projisere flere 3D objekter samtidig enten ved hjelp av perspektivisk- eller isometrisk projeksjon, men den vil ikke takle flere avanserte objekter som for eksempel hyperbolsk parabol. Andre avanserte 3D objekter er flater generert ut fra cubic Beizer kurver. Videre er det mulig å rotere, enten med quaternion eller matriser, skalere og transelere objektene ved bruk av mus. Interaksjonen med scrollbarer og tekstbokser eksisterer ikke i

brukergrensesnittet, men metodene for dette ligger klare i kildekoden til 3D modulen. Avanserte objekter kan projiseres med rutenett, en farge, flere farger, gradient fordelt farge i rutene eller gråtoner. Rutenettet kan tegnes opp sammen med de andre alternativene. I tillegg er det implementert inn en metode for å beregne normalvektoren for en eller flere ruter i et avansert objekt.



Figur 9.1 3D modulen implementert i parAbels syntetiske kalkulator

10 DRØFTING

Det har kommet frem at isometrisk projeksjon vil være den beste visualiseringsmetoden for 3D modulen, på grunn av at parallelle linjer har lik forkortelse, når de blir visualisert. Denne egenskapen ble tatt i betraktning siden 3D modulen skal implementeres i en syntetisk kalkulator og gi objekter en best mulig matematisk fremstilling. Den første hypotesen som ble presentert gikk ut på at en perspektivisk projeksjon ville gi en dybdeføelse som er mer tilnærmet det menneskelige øyet og vil dermed gi et mer virkelig bilde av 3D objektet. Det at en perspektivisk projeksjon gir et med virkelig bilde av 3D objektet vil ikke være sant, sett fra et matematisk synspunkt. Meningen er at objektet skal fremstilles mest mulig korrekt, i forhold til ligningen som har generert objektet. Dette vil si at en isometrisk projeksjon vil gi et mer matematisk korrekt bilde av 3D objektet enn perspektivisk projeksjon.

ActionScript har to metoder for opptegning av objekter. curveTo() metoden egner seg kun til spesielle kurver som ikke interpolerer kontrollpunktet til kurven. Metoden ble også testet i forbindelse med bruk av midtpunktsalgoritmen for generering av cubic Bezier kurver. Her kom det blant annet frem at midtpunktsalgoritmen kun egner seg til ren visualisering av en cubic Bezier og ikke dersom det skal benyttes matematiske metoder. Den beste løsningen å generere en cubic Bezier kurve på, vil være å benytte ligningen til kurven. Siden 3D modulen er lagt opp til å visualisere matematiske uttrykk, vil curveTo() metoden aldri bli brukt. Den andre opptegningsmetoden, lineTo(), tegner et rett linjesegment med en gitt lengde. Dette fører til at metoden kan benyttes til å representere alle typer objekter. Bakdelen med denne metoden er at det kreves mange linjesegmenter dersom et objekt skal representeres med godkjent kvalitet. Når antall linjesegmenter økte, fikk ActionScript problemer med kalkuleringstiden. lineTo() vil allikevel være den eneste metoden som kan representere objekter korrekt og må derfor benyttes. Eventuelle forsinkelser som denne metoden gir, når den blir benyttet mange ganger, vil være en begrensing i ActionScript.

Cubic Bezier kurver har to kontrollpunkter og gjør det derfor enklere å generere egendefinerte flater. Opp mot hurtighet vil ikke cubic Bezier være beste løsningen, siden curveTo() metoden har samme egenskaper som en quadratic Bezier kurve. Forsinkelsen av generering av kurver er derimot så liten at brukervennligheten vil komme foran.

Det har blitt diskutert to mulige løsninger for å tegne opp linjene en flate består av. Den første og raskeste av metodene tegner opp kurvelinje for kurvelinje og disse linjene danner et rutenett. Den andre metoden gikk ut på å tegne rute for rute. Det kom frem i resultatene at rute for rute opptegning var nødvendig dersom flaten skal bestå av flere farger og for å gjøre matematiske utregninger på flaten i etterkant raskere. I tillegg tegnet rute for rute metoden opp flere linjer to ganger som gir flere punkter. Dersom en flate ikke skal representeres med farger vil det allikevel være mulig å finne rutene i et rutenett, men dette krever flere beregninger. Et annet resultat som kom frem ved rotasjon av objekter var at ActionScript brukte lang tid på å kalkulere rotasjonen til punktene. Siden kurvelinje for kurvelinje gir færre punkter, kan det være mer hensiktsmessig å benytte punktene denne metoden genererer til rotasjon, skalering og translasjon. De sistnevnte metodene vil da gå raskere, men det vil ta lenger tid å gjøre beregninger på flaten innenfor en rute. Det vil med andre ord være et valg om det er ønskelig med en mindre forsinkelse i alle operasjoner eller om noen operasjoner ikke skal ha forsinkelse, mens andre har stor forsinkelse. Dette gjelder selvfølgelig kun dersom flaten ikke skal bli fylt med farger. Et eksempel på dette er visning av normalvektor. Det kom frem at det var mest hensiktmessig å benytte rutenettet til å velge hvor normalvektoren skal vises. Hvilken opptegningsmetode som blir benyttet for å tegne dette rutenettet har ingen innvirkning på valg av rute. Hvis rute for rute opptegning blir benyttet, vil informasjon om alle rutene bli tatt vare på. Det vil da gå raskt å få tak i ruten som er valgt og finne hvilke verdier som skal benyttes, men andre operasjoner som rotering, skalering og translasjon vil gå tregere. Dersom kurvelinje for kurvelinje blir benyttet vil rotasjon, skalering og translasjon gå raskere, men det vil ta lengre tid å regne ut hvilken rute som er valgt og hvilke verdier som skal benyttes til normalvektoren. I rapporten ble det utledet hvordan antall punkter en flate består av kan optimaliseres. Denne optimaliseringen viste at det var mulig å oppnå en god kvalitet på flaten samtidig som forsinkelsen ble redusert betraktelig. Bruk av optimalisert rute for rute opptegning vil da ikke inneholde store forsinkelser og samtidig holde på informasjon om alle rutene.

I tillegg vil denne metoden ikke legge begrensinger i forhold til fargelegging.

Det har kommet frem at en flate kan bli representert ved hjelp av flere fargemønster. Rutenett er den enkleste formen for representasjon av en flate. Rutenettet bygger opp rammene til flaten og gir den et romlig bilde. Uten dette rutenettet i bunnen vil flaten miste 3D perspektivet ved bruk av farger. Det som gir ActionScript en fordel i fargelegging er gradient metoden. Fargelegging tar tid og ActionScript sliter med hurtigheten allerede uten farger. Gradient kan da representere flaten på en god måte uten at det ligger et detaljert rutenett i bunnen. Bakdelen med gradienten er at den krever mye informasjon som kan være vanskelig å få tak i. Det har blitt jobbet veldig mye med gradient i dette prosjektet og hvordan informasjonen som trengs for å beregne retningen til gradienten kan bli hentet ut. Eksempelet som ble brukt i denne rapporten var farger basert på y verdi til rutene. Dette er et enkelt eksempel, men det var allikevel vanskelig å få tak i informasjonen som var nødvendig for å bestemme retningen til gradienten. Gradient er et veldig kraftig verktøy og kan ha stor innvirkning på hurtigheten og effektiviteten til en 3D modul. Bruken av gradient metoden må ses opp mot fargefordelingsmetoden som blir benyttet. Hvis denne metoden er for avansert og gradienten ikke viser korrekt resultat i enkelte posisjoner, bør metoden unngås.

Interaksjonsmuligheter er en av de viktigste oppgavene til 3D modulen utenom selve visualiseringen av objektene. Hvilke interaksjonsmuligheter som eventuelt er de beste er vanskelig å fastslå siden alle har sin egen mening om hva som faller mest naturlig. Det har allikevel blitt diskutert ulike måter å interaktere på og hvilke begrensninger som kan oppstå med de ulike hjelpemidlene. Resultatet av denne diskusjonen er at siden det eksisterer ulike begrensninger, for eksempel ved bruk av mus, vil det være mest hensiktsmessig å benytte en kombinasjon av flere hjelpemidler. Hvordan det er ønskelig informasjon skal vises kan variere fra person til person. Eksempel på dette er om aksene til koordinatsystemet skal skjære/gå igjennom objektet eller om dette skal vises for seg selv utenfor objektet. Denne informasjonen må uansett vises på en eller annen måte, siden orienteringen av aksene kan være forskjellig fra system til system. I denne 3D modulen er det valgt et oppsett x aksen har retning mot høyre, y aksen har retning oppover og z aksen innover i skjermen. Dette systemet har blitt valgt med grunnlag i at det gir en enkel overgang fra et 2D orientert system.

De to rotasjonsmulighetene som har blitt testet opp mot hverandre i denne

oppgaven er quaternion og bruk av matriser. Den andre hypotesen som ble presentert i begynnelsen av rapporten gikk ut på at quaternion rotasjonen egner seg bedre enn matrise rotasjon i ActionScript. Quaterenion viste seg å være mer effektiv enn matrise rotasjon, når disse ble testet opp mot hverandre. I forhold til ActionScript vil hypotesen over være sann, både på grunn av at quaternion viste seg å rotere objekter raskere enn matriser, men også på grunnlag av at ActionScript allerede er presset i forhold til jevn rotasjon. Derfor vil quaternion ha større innvirkning på et språk som sliter med 3D objekter, som ActionScript, i forhold til språk som er raskere på 3D.

Det er i denne rapporten diskutert ulike måter å rotere et objekt på. Det har blitt trukket frem globale koordinatsystemer som gjelder for alle objektene og lokale koordinatsystemer som gjelder for hvert enkelt objekt. Resultatet av diskusjonen rundt disse rotasjonsmetodene var at det burde være mulig å rotere objektet rundt et lokalt koordinatsystem i tillegg til det globale koordinatsystemet. Grunnet til dette er at objektet kan ligge langt unna origo til det globale koordinatsystemet. Dersom det kun var mulig å rotere rundt det globale koordinatsystemet ville objektet flyttet seg med lik avstand i forhold til origo. Så for at en bruker skal slippe å flytte et objekt ned til origo kun for å rotere rundt en akse som skjærer objektet, bør objektet ha et eget lokalt koordinatsystem. Dermed kan objektet roteres rundt et aksesystem som er definert der objektet er tegnet opp.

For at objektene skal kunne roteres må alle punktene til objektene bli lagret. Denne modulen lagrer all informasjon om objektet i en array liste. Denne formen for lagring gjør at modulen får rask tilgang til informasjonen om objektet. Informasjonen blir ikke lagret etter modulen har blitt avsluttet. Det vil allikevel være fult mulig å lagre informasjonen permanent i eksterne filer, selv om denne muligheten ikke er blitt implementert i modulen. For å unngå inkonsistens i flere ulike lagringslister, ble det benyttet Singleton Design Pattern. Denne strukturen løste problemstillingen rundt lagring av flere objekter.

Alle resultatene som har blitt presentert er implementert i en prototyp modul. Denne 3D modulen genererer, håndterer og visualiserer 3D objekter. Ikke alle interaksjonsmulighetene er implementert og dette kommer av at prototypen er en modul som skal implementeres i en annen applikasjon som håndterer brukergrensesnittet. Det er derfor bare testet om det er mulig å få tak i verdiene som trengs for å benytte ulike interaksjonsmuligheter. Noen interaksjonsmuligheter som

eksempelvis rotasjon med mus er allikevel implementert fordi den krever egne metoder. Denne modulen vil være et utgangspunkt for videre utvikling. Den inneholder eksempler som viser hvor informasjon skal sendes og hvordan informasjonen skal behandles. Eksempel på dette er fargefylling basert på høyde. Denne metoden kan brukes som utgangspunkt for andre fargefordelingsmodeller. Lagringsstrukturen for en hyperbolsk parabol kan benyttes på alle typer flater. Neste steg for denne modulen vil være å tolke det matematiske uttrykket gitt fra kalkulatoren og sende det til klassen som tegner opp flaten. Modulen har blitt implementert og testet sammen med den syntetiske kalkulatoren som har vært utgangspunktet for denne oppgaven. Implementasjonen gikk uten problemer og all funksjonalitet som var utviklet i modulen virket.

11 KONKLUSJON OG VIDERE ARBEID

11.1 KONKLUSJON

I denne diplomoppgaven har det blitt utviklet en prototyp av en 3D modul som genererer, håndterer og visualiserer 3D objekter. Den er utviklet ved hjelp av ActionScript 2.0 og implementert i parAbels syntetiske kalkulator. Gjennom hele rapportperioden har de ulike delene som 3D modulen består av, blitt evaluert med tanke på om de egner seg i en slik 3D modul. Siden kalkulatoren skal benyttes i undervisnings sammenheng var det nødvendig å evaluere hvilke visualiserings teknikker og interaksjons muligheter som var mulig å implementere og hvilke av disse som egnet seg best for læring.

Perspektivisk projeksjon egner seg kun for visualisering, mens isometrisk projeksjon egner seg for visualisering hvor det skal gjøres matematiske operasjoner på det projiserte 3D objektet.

3D modulen gir brukeren mulighet til å interaktere med mus, scrollbarer og tekstbokser på de visualiserte 3D objektene. Dette gir mulighet for å tilpasse 3D modulen til brukere med ulikt kunnskapsnivå.

Det er mulig å generere og projisere detaljrike 3D objekter som det kan utføres matematiske beregninger på. Skal det projiserte 3D objektet transformeres over tid, er det nødvendig å optimalisere utregningen av 3D objektene. Optimaliseringen vil redusere detaljrikheten.

Arbeidet med 3D modulen har vist at ActionScript får store problemer når antall kalkulasjoner øker. Resultatet av dette gjør at 3D modulen kun har mulighet til å håndtere ett detaljrikt 3D objekt om gangen.

11.2 VIDERE ARBEID

Det vil være nødvendig å implementere flere matematiske beregninger på det projiserte 3D objektet, da prototypen av 3D modulen kun inneholder en slik matematisk beregning, Normalvektoren.

En metode som tolker matematiske uttrykk fra kalkulatoren er ikke laget og må implementeres. Til nå har de matematiske uttrykkene blitt hardkodet inn i kildekoden.

12 REFERANSER

[1] www.nvu.no/nvukonferansen2006/Olav-Aas.doc?PHPSESSID=f14a7fb5796d00d64d6df938585cd0d8 (01.05.2006)

[2] http://www.parabel.no/om_parabel.htm#bakgrunn (01.05.2006)

[3] http://en.wikipedia.org/wiki/3D_projection (13.12.2005)

[4] http://www.cmspro.com/3d_graphics.htm (13.12.2005)

[5] http://en.wikipedia.org/wiki/Isometric_projection (14.12.2005)

[6] Foley, Van Dam, Feiner, Hughes og Phillips (1999) *Introduction to computer graphics.*

[7] Gordon, Dan (2002) *The Floating Column Algorithm for Shaded, Parallel Display of Fuction Surfaces without Patches* http://ieeexplore.ieee.org/iel5/2945/21152/00981853.pdf?tp=&arnumber=981853&isnumber=21152 (05.11.2005)

[8] Tankus, A, Sochen, N, Yeshurun,Y (2003) *A New Perspective [On] Shape-from-Shading* http://lear.inrialpes.fr/people/triggs/events/iccv03/cdrom/iccv03/0862_tankus.pdf (05.11.2005)

[9] Permuter, H, Francos, J M (2000) *Estimating the Orientation of Planar Surfaces: Algorithm and Bounds* http://www.ee.bgu.ac.il/~francos/perscrbit_final.pdf (03.11.2005)

[10] Permuter, H, Francos, J M (2001) *Parametric Estimation of the Orientation of Textured Planar Surfaces* http://www.ee.bgu.ac.il/~francos/persip.pdf (25.11.2005)

[11] Elad, A, Kimmel, R (2003) *On Bending Invariant Representation for Surfaces Reconstruction of 3D objects from Three Orthographic Views* http://www.cs.technion.ac.il/~ron/PAPERS/IEEE_PAMI_2003.pdf (25.11.2005)

[12] Liu, S, Hu, S, Tai, C, Sun, J (2000) *A Matrix-Based Approach to Reconstruction of 3D Objects from Three Orthographic Views* http://ieeexplore.ieee.org/iel5/7096/19120/00883948.pdf?arnumber=883948 (23.11.2005)

[13] Migeon, B, Rosenberger, C, Marche, P (2001) *Correction of zoomed morphology-based interpolation of contours* http://ieeexplore.ieee.org/iel5/7934/21882/01017294.pdf?arnumber=1017294 (27.11.2005)

[14] Lay, D C (2003) *Linear Algebra And Its Applications*

[15] http://en.wikipedia.org/wiki/Transformation_matrix (14.12.2005)

[16] Karagianni, K, Stouraitis, T (2001) *A Vector Processor For 3-D Geometrical Transformations* http://ieeexplore.ieee.org/iel5/7344/19936/00922279.pdf?arnumber=922279 (27.11.2005)

[17] Thevenaz, P, Unser, M (1995) *Efficient Geometric Transformations And 3-D Image Registration* http://bigwww.epfl.ch/publications/thevenaz9502.pdf (27.11.2005)

[18] http://en.wikipedia.org/wiki/Quaternion (14.12.2005)

[19] Hoffmann, G (1978) *Application of Quaternions* http://www.fho-emden.de/~hoffmann/quater12012002.pdf (28.11.2005)

[20] http://astronomy.swin.edu.au/~pbourke/fractals/quatjulia/ (14.12.2005)

[21] Kanatani, K (1994) *Analysis of 3-D rotation Fitting* http://ieeexplore.ieee.org/iel1/34/7215/00291441.pdf?tp=&arnumber=291441&isnumber=7215 (28.11.2005)

[22] http://www.macromedia.com/ (14.12.2005)

[23] http://www.actionscript.org/tutorials.shtml (13.12.2005)

[24] http://webpages.charter.net/gubber/flash/3DEnginel.swf (13.12.2005)

[25] Livingston, Dan (2005) *ActionScript 2.0 Garage*

[26] http://en.wikipedia.org/wiki/Bézier_curve (05.04.2006)

[27] http://www.timotheegroleau.com/Flash/articles/cubic_bezier_in_flash.htm (06.05.2006)

[28] http://en.wikipedia.org/wiki/Singleton_pattern (16.04.2006)

[29] Edwards, C, Penney, D (1988) *Elementary Linear Algebra*

[30] Edwards, C, Penney, D (1998) *Calculus with analytic geometry*

[31] http://www.sjbrown.co.uk/?article=quaternions (15.05.2006)

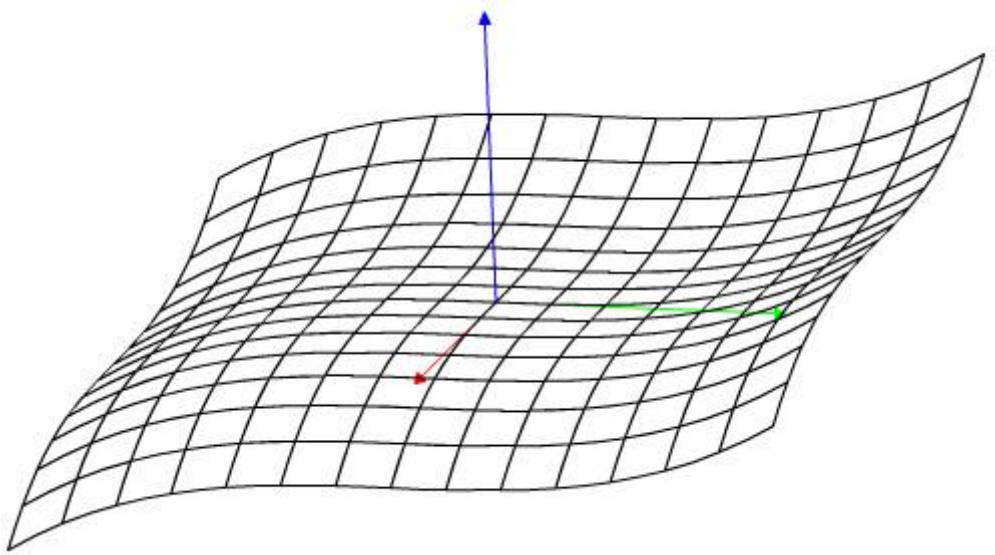
[32] http://www.adobe.com/devnet/flash/articles/3d_classes_03.html (15.05.2006)

[33] http://www.anticz.com/eularqua.htm (15.05.2006)

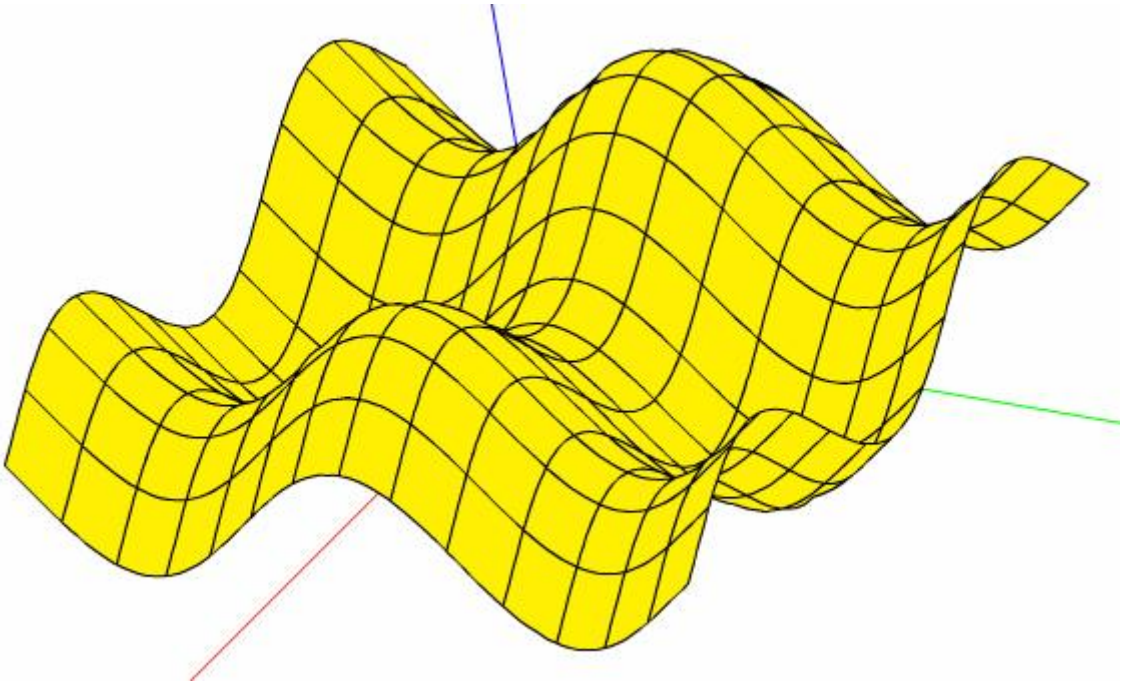
[34] http://en.wikipedia.org/wiki/Right_hand_rule (12.05.2006)

[35] http://www.matematikk.net/ressurser/per/per_oppslag.php?aid=240 (23.04.2006)

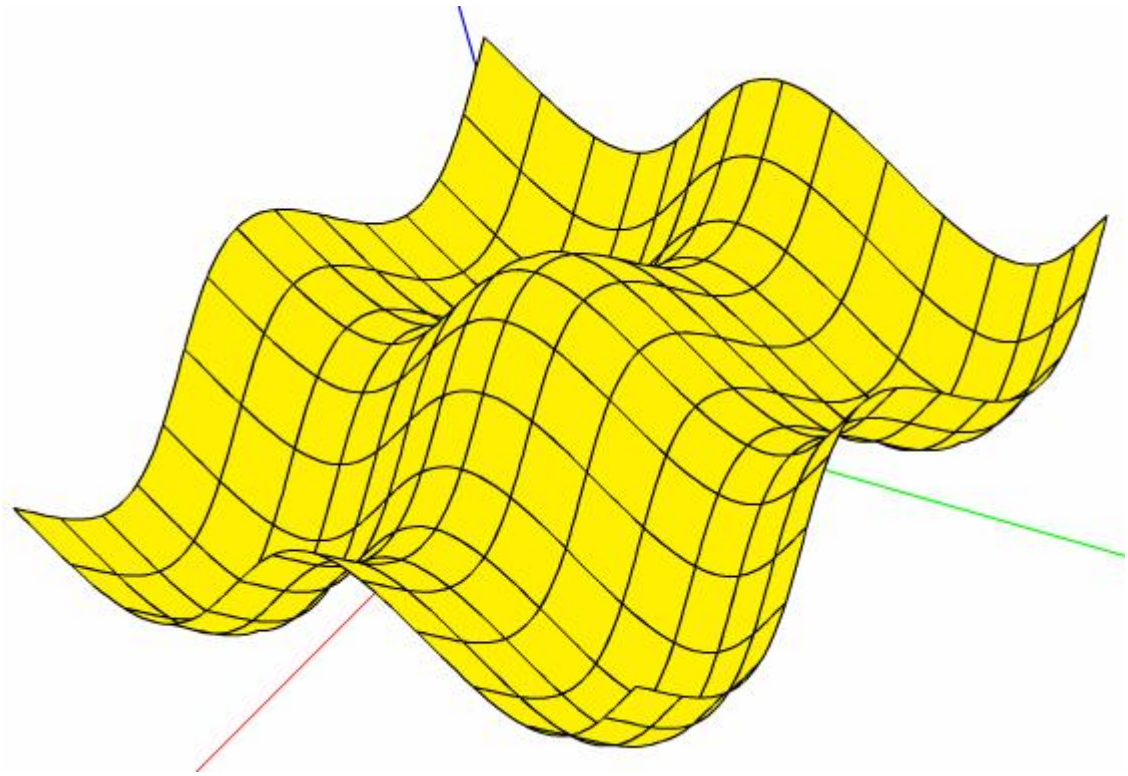
VEDLEGG 1



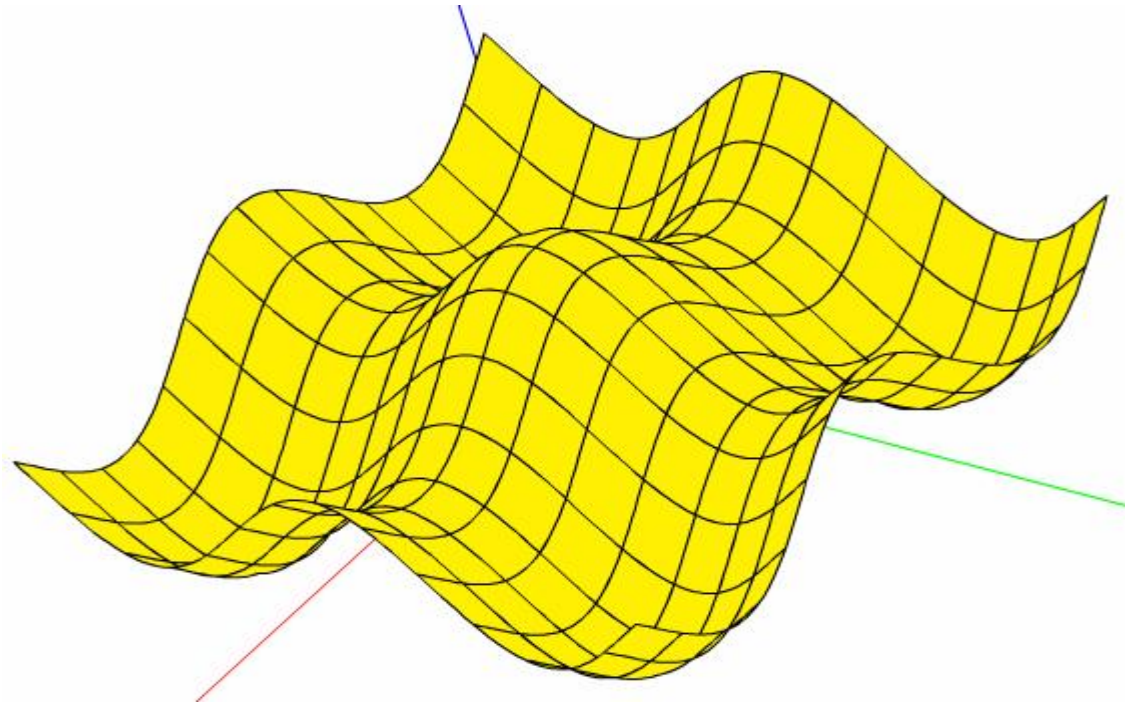
Figur: $y = x^3 - z^3$



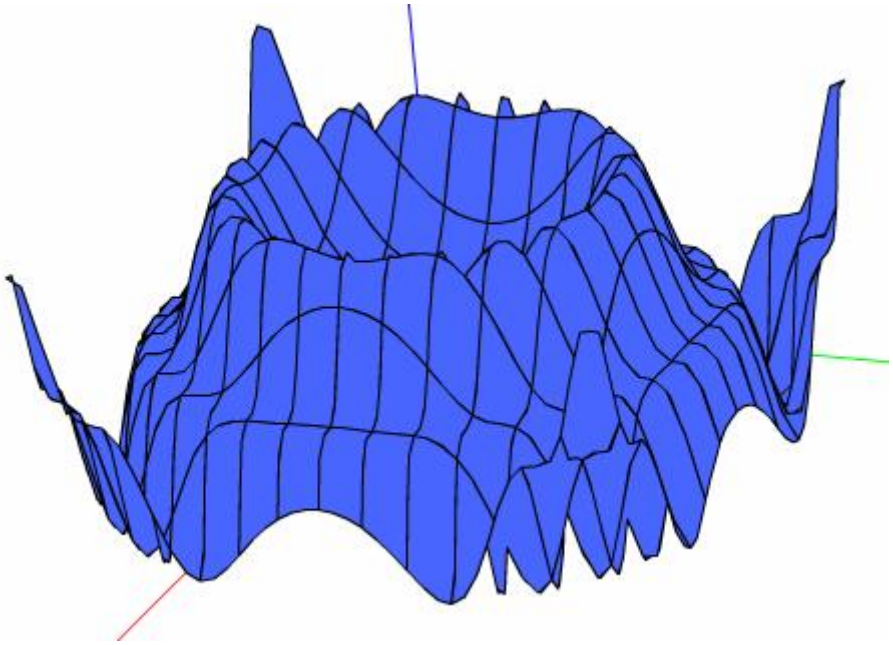
Figur $y = \cos(x) - \cos(z)$



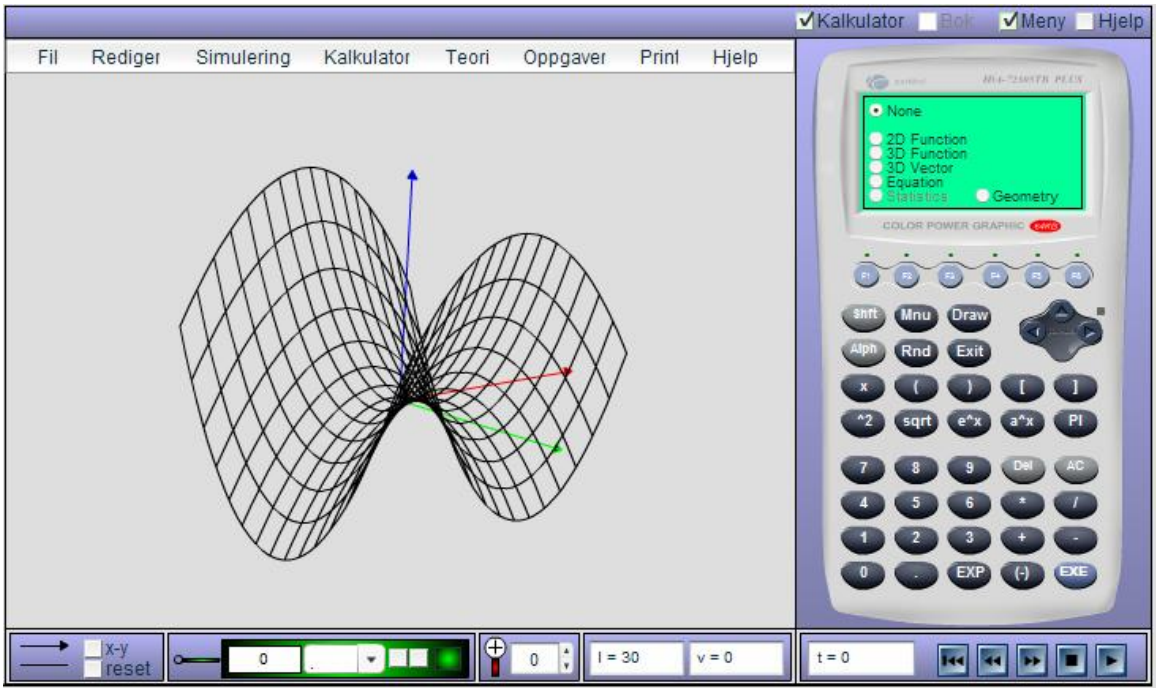
Figur: $y = \cos(x) + \cos(z)$



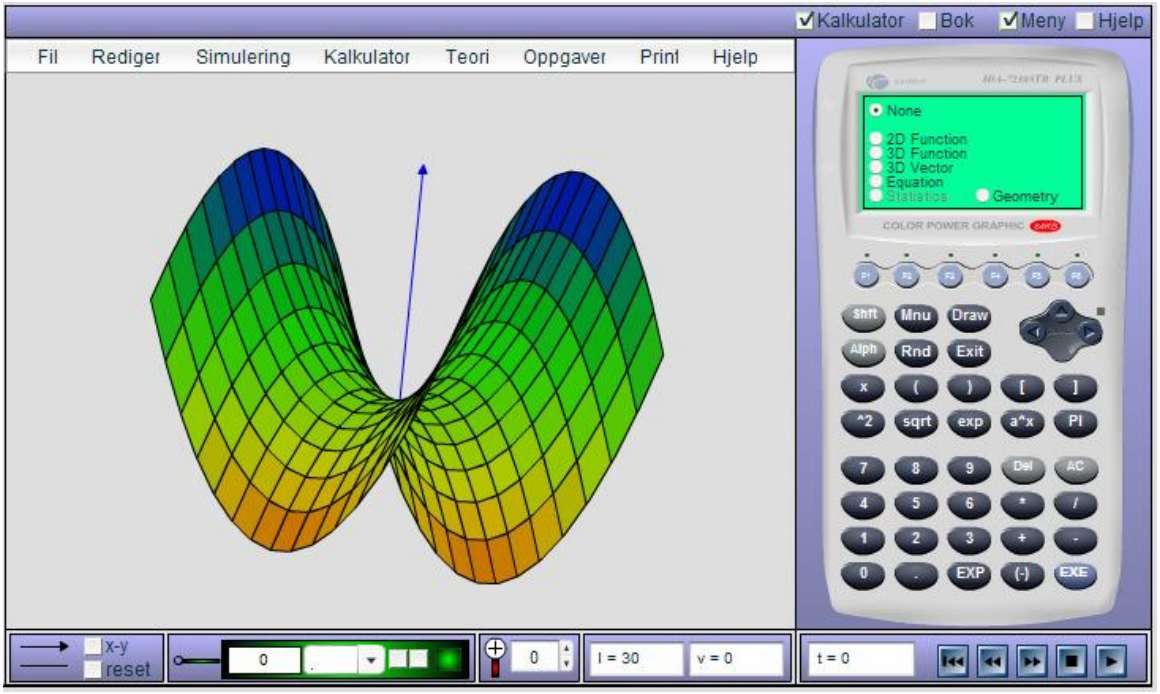
Figur: $y = |\cos(x) + \cos(z)|$



Figur: $y = \left(\sin(x^2 + z^2)\right)^2$



Figur: Rutenett av en hyperbolsk parabol vist i parAbels syntetiske kalkulator



Figur: Hyperbolsk parabol med flere farger vist i parAbels syntetiske kalkulator