Software Tools in Industrial Mathematics

Working Notes #9

GraphView - En Java og CGI–pakke for 3D grafikk

Åsmund Ødegård
Gerhard W. Zumbusch

9. september 1996

Innhold

GraphView - En Java og CGI–pakke for 3D grafikk

Åsmund Ødegård
Gerhard W. Zumbusch

1 Introduksjon

I våre dager er det viktig å profilere seg gjennom Internett. En av de siste nyvinningene i denne forbindelse, er muligheten for å legge inn applikasjoner på Web–sider, som gir mulighet for interaksjon mellom Websiden og publikum. Dette har tidligere vært mulig ved hjelp av CGI, The Common Gateway Interface. Ved hjelp av denne teknologien kan man kjøre script, programmer etc. på Server, det vil si "hos seg selv," relativi til hvem som har laget Websiden. Ved hjelp av slike verktøy, kan man hente ut informasjon, lage datasett ved hjelp av programmer ol. Den siste nyskapningen er imidlertid Java. Dette er et programeringsspråk spesialdesignet for å ha en dynamisk interaksjon med publikum på Web–sider. I motsetning til CGI, vil et Java program kjøre hos klient, det vil si lokalt hos den som ser på Web–siden. Man er dermed ikke begrenset av at data skal sendes over nettet. Dette gjør det for eksempel mulig å ha mus–interaksjon på Websider. Bakdelen er at man ikke kan ha optimalisert kode, med hensyn på arkitektur, siden koden skal kunne kjøres på mange forskjellige arkitekturer. Koden som kjøres er derfor "halv–kompilert", slik at den må tolkes endelig av Web–brouseren hos klienten. Dette gjør at hastigheten ikke er svært stor, men absolutt brukbar.

Hva har så alt dette med "GraphView" å gjøre. Det vi har gjort er å kombinere CGI og Java. Vi har laget et Java program, som tar seg av grafikken. Videre bruker vi CGI–script for å fremskaffe dataene som vi vil vise.

2 GraphView

GraphView er en såkalt Java–applet. Dette betyr at det er et program som ikke er ment å kjøre som en selvstendig applikasjon, men i et annet program, først og fremst WWW–programmer. Etterhvert finnes det en del slike programmer som støtter Java, for eksempel versjoner av Netscape etter 2.01, og så vidt jeg vet, de siste versjonene av Microsoft Explorer (for de som er Microsoft–fantaster. . .). For å kunne kjøre programmet, skriver man litt HTML–kode, hvor man legger inn et <APPLET>–merke. Når WWW–programmet leser dette, vil programmet starte automatisk.

2.1 Hvordan bruke programmet

Alt man trenger å gjøre for å bruke programmet, er å legge inn følgende lille bit HTML–kode på Web–siden sin:

```
<APPLET
      name= “GraphView”
      code=  “GraphView.class”
      width= “400”
      height=“500”
      align=  “Top”
>
<param
      name= “scale”
      value= “1”>
      <hr> If your browser recognized the applet tag,
      you would see an applet here.<hr>
</APPLET>
```

For at dette skal fungere, må filen med HTML–koden og den kompilerte Java–koden ligge i samme filkatalog. “Namer” er et navn på appleten, det er valgfritt om man vil legge inn et navn for dette programemt. “Code” angir navnet på filen som inneholder hoveddelen av programmet, som i dette tilfellet er “GraphView.class”. Dette vil for en applet alltid være et filnavn som har tillegget “.class.” Dette henspeiler på at Java er et Objektorientert språk, der ingen kode eksisterer utenfor klasser. Når man kompilerer, vil være kompilerte klasse legges i en egen fil, med navnet “klassenavn.class.” Den klassen vi ønsker å kalle i “code” er den kjørbare klassen, som i vårt tilfelle er “GraphView.class.” For at det skal være mulig å kjøre programmet må følgende klasse–filer finnes i samme katalog: “Matrix3D.class,”

"Model3D.class" og "FileFormatException.class." "Height" og "width" er selvforklarende, verdiene er i pix'. Verdiene som angis relateres til grafikk–vinduet som vises i WWW–programmet når appleten kjøres. "Align" brukes til å plassere appleten på Web–siden, lovlige verdier er "top", "middle" og "bottom."

Mellom <APPLET> og </APPLET> kan man angi parametere til programmet. I vårt tilfelle er det to lovlige parametere, det er "scale" og "script". "Scale" brukes til en initiell skalering av grafikken som appleten viser. Verdien "1" gir ingen skalering, større verdier skalerer opp og mindre verdier skalerer ned. Parameteren "script" benyttes til å angi hvilket CGI–script som skal kjøres for å fremskaffe nødvendig data til programmet. Jeg skal senere komme nærmere inn på hva scriptet må inneholde. Det er frivillig å angi scriptets navn, default–verdien er satt til `http://www.oslo.sintef.no/cgi-bin/diffpack/diffpack-demo/Graphview`.

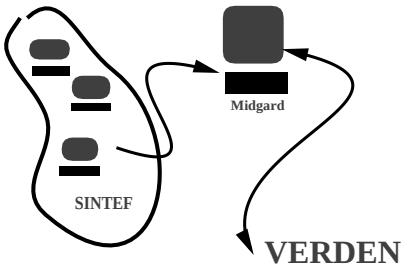
Tilslutt er det skrevet en tekst. Denne vil vises på skjermen i de tilfellene hvor brukeren har et WWW–programm som ikke kan vise Java–kode.

Hvis den kompilerte koden for Java–programmet ligger i en annen filkatalog en den HTML–koden ligger i, må man sette inn attributten "codebase" i <APPLET>–merket. Denne atributten skrives som en URL–adresse, det vil si på formen `http://www.../am/...`, hvor man spesifiserer adressen frem til filkatalogen der koden er plassert.

Dette er essensielt alt man trenger å vite for å kunne kjøre programmet, med det eksemplet som foreligger i dag. Imidlertid vil det være aktuelt å forandre på programmet, derfor vil jeg beskrive det hele i mere detalj.

3 CGI–Script Interaksjon med Applet

Før programmet "GraphView" kan plotte noe som helst, må det foreligge et datasett som man ønsker å visualisere. Når programmet starter opp, vil det kalle et script, med default verdi som angitt over, eller som angitt i parameteren "script." Deretter vil programmet vente på data. "GraphView" vil dessuten forutsette at de dataene som kommer er på "plotMTV"–format, slik dette ser ut for regulære gitter, det vil si med jevn oppdeling. Vi må derfor skrive et script som returnerer slike data, for å kunne benytte "GraphView." Dette kan gjøres på mange måter, men jeg vil i det følgende bare se på det tilfelle som er aktuelt for vårt mål med å lage "GraphView," nemlig å kunne presentere Diffpack på en ny og spennende måte på Web–sidene til diffpack–prosjektet ved SINTEF–Anvendt Matematikk. Denne synsvinkelen gjør at vi vil ta opp flere aspekter enn bare CGI–scriptet, som er spesielt relatert til data–nettverket på SINTEF.



Figur 1: SINTEF's nettverk, sett utifra http–synsvinkel.

3.1 Diffpack–Applikasjoner og Ekstern Web–server

Vårt hovedmål er å kunne starte en diffpack–aplikasjon, det vil si en likningsløser basert på diffpack–bibliotekene, fra CGI–scriptet vårt. For å kunne gjøre dette er det imidlertid nødvendig at diffpack–applikasjonen er kompilert for å kunne kjøre på SINTEF'S eksterne server, midgard. Grunnen til dette er at det bare er filer plassert på midgard som kan hentes ned fra maskiner utenfor SINTEF, mer teknisk sagt er nettet på SINTEF omgitt av en "Firewall". Dette gjør at det heller ikke er mulig å hente filer eller kjøre programmer plassert på andre servere, fra midgard, som er utenfor brannveggen, se figur 1 for en grafisk fremstilling av problemet.

Hovedproblemet i denne sammenheng, er at det ikke er meningen at midgard skal være en server man arbeider på til vanlig. Det er derfor lagt opp svært lite verktøy og programmvare på denne serveren. Blant annet er det ikke installert kompilatorer på serveren. Systembibliotekene som man vanligvis linker dynamisk når man bygger en applikasjon, finnes heller ikke på midgard. Dermed må man kompilere applikasjonen statisk, slik at alle nødvendige bibloteksrutiner legges inn i den eksekverbare filen. Deretter kan man flytte applikasjonen til et filområde på midgard, slik at dataene som produseres kan visualiseres med "GraphView."

For å kompilere slik at man kan kjøre en diffpack–applikasjon på midgard må man legge til noen linjer i file .pdmkvars, som leses når man make'er en applikasjon. Følgende magi er det nødvendige:

- LDOPT += -Wl,-a,archive
- SYSLIBS := -Wl,-a,default -lcl -Wl,-a,archive $(SYSLIBS)

Hvis man velger å kompilere uten å bruke "make," er man overlatt til seg selv, men da er man vel også så flink at man klarer å ekstrahere nødvenig viten ut av linjen over. Det man essensielt gjør, er å legge til "-Wl,-a,archive" helt i starten, slik at alt linkes statisk. I tillegg må man linke noen system biblioteker spesielt, dette gjøres ved å legge til "-lcl" før de andre systembibliotekene, pluss noe magi!. Man har nå en applikasjon som kan kjøres på midgard, forutsatt at man husket å kompilere på

en HP–maskin, som for eksempel verdande eller ydale, siden midgard er en HP-maskin.

3.2 Diffpack–applikasjon og CGI

Nå når vi har en ferdig kompilert, og kjørbar applikasjon, kan vi lage et CGI script, som kan kalles av WWW programmet, eller snarere av httpd, http–demonen som kjører på SINTEF's eksterne Web–server. Scriptet må legges under filområdet `/SI/WWW/http/cgi/diffpack`. Men her er det bare drift som kan legge inn noe som helst. Det er derfor laget til et annet filområde på den eksterne serveren, `/SI/WWW/http/scripts/diffpack` hvor vi kan legge inn script. Det er imidlertid bare script, eller snarere script navn satt opp av drift som vil være akseserbare. Det må nemlig ligge et program under `/SI/WWW/http/cgi/diffpack` som kan kalle scriptet i katalogen `/SI/WWW/http/scripts/diffpack`. Slik det er nå, vil bare scriptet lokalisert her, under navnet `diffpack-demo`, være akseserbart som et CGI script, det vil si ved hjelp av URL'en `http://www.oslo.sintef.no/cgi-bin/diffpack/diffpack-demo`.

For å komme videre, er det mange måter å gå frem på. Den første, og enkleste er å alltid nøye seg med et script, som igejn vil føre til at vi bare kan kjøre en applikasjon, slik som systemet foreligger idag. Den nest enkleste løsningen vil være å springe til drift vær gang man trenger et nytt script. Vanskelighetsgraden av dette vil avhenge sterkt av arbeidspress og annet på drift, men det er selvfølgelig en god løsning for den som ikke liker bakveier. . . Det finnes en siste mulighet, som er den jeg har benyttet hittil, som gir oss mulighet til å kalle flere script gjennom det ene "offisielle" cgi–scriptet. Før vi forklarer hvordan dette gjøres må vi se litt generelt på hvordan CGI virker.

Når man kaller et cgi–script har man mulighet til å bruke to forskjellige metoder for å sende data til scriptet, under navnen "GET" og "POST." For metoden "GET" sendes dataene ved hjelp av URL'en som kaller scriptet. Det finnes igjen to forskjellige måter å gjøre dette på. Den første måten er å legge til en slash, og deretter en vilkårlig string til URL'en, for eksempel slik:

```
http://www.oslo.../cgi-bin/.../diffpack-demo/GraphView
```

Hvis det ikke finnes en filkatalog under diffpack med navnet diffpack-demo, vil http–protokollen, som tolker URL'en legge til varabelen "PATH_INFO" med verdien "GraphView" i omgivelses–varaiblene ("environment") til scriptet "diffpack-demo" Deretter vil scriptet bli kjørt. Den andre muligheten for å sende input til scriptet med denne metoden, er å legge til et spørsmålstegn, og deretter "navn=verdi" par adskilt av &–tegnet. Det vil se slik ut:

```
http://..../cgi-bin/diffpack/diffpack-demo?navn1=verdi1
&navn2=verdi2
```

Disse parene blir lagt inn i scriptets omgivelses–variable, som "QUERY_STRING." De forskjellige parene blir adskilt av +–tegn. Man kan deretter dekode denne variabelen, hvis man ønsker dette.

Den andre metoden man kan benytte for å sende data fra en Web–side til et cgi–script, er å bruke "POST" metoden. Denne brukes i forbindelse med sending av data fra såkalte "forms" på Web–sider. Dette skal vi ikke gå nærmere inn på foreløpig, siden vi ikke benytter dette i systemet. I midlertid kan man gjerne tenke seg muligheten av å ha mulighet for å sette parametere direkte på Web–siden, ved hjelp av "forms", og benytte "POST" metoden, slik at fremtidige vedlikeholdere og utviklere av systemet bør sette seg inn i de mulighetene som finnes. Det viktige nå er at det som er sagt over om "PATH_INFO" fungerer på samme måte for "POST."

For den omtenksomme leser burde det nå være enkelt å se hvordan man skal få det ene scriptet til å fungere som flere. Ganske enkelt ved å bruke "PATH_INFO." Denne variabelen kan man sjekke på, og deretter kalle et pessende script utifra verdien i "PATH_INFO." Det er også noe mer komfortabelt at scriptet som nå kalles, kan ligge på vilkårlig sted på midgard. Rent praktisk har vi latt `diffpack-demo` være et *perl* script, som består av en lang if–test, hvor man stadig kan sette til nye lovlige verdier for "PATH_INFO" og dertil hørende script som skal kalles. Disse kalles med kommandoen "exec" som kjører kommandoen som gis i et vanlig *shell*.

Vi må nå se på hva et script må inneholde. I sin enkleste form kan et script bestå av noen få linjer. Det må kjøres en applikasjon, med passende input, deretter må vi kjøre "simres2mtv," som også må være kompilert til å kjøre på midgard[1]. Tilslutt må datasettet som er laget, sendes til standard output. Når scriptet er kalt ved hjelp av et cgi–script, vil standard output være satt opp av httpd, slik at data'ene sendes direkte tilbake til klienten som kom med forespørsel om den aktuelle URL'en. I tillegg til dette, må vi først sende noen linjer som tolkes av klienten, for at denne skal vite hva slags data som kommer. Et lite script–eksempel kan se slik ut:

```
#!/bin/sh

app < input.i
simres2mtv -f SIMULATION -s -n u -a
#START OUTPUT OF DATA
echo "200 OK"
echo "Content-type: text/plain"
echo ""
cat SIMULATION.scalar.mtv
```

Dette er i hovedprinsippene hva vi trenger, men i praksis må scriptet modifiseres noe. For det første vil GraphView sende med to "navn=verdi" par når det kaller

[1] Vanligvis må man besørge denne kompileringen selv, men i skrivende stund finnes det et eksemplar på området `/users/aod/Scripts/DP-app/s2mtv` på midgard

scriptet. Det som sendes er "domain= $[a, b] \times [c, d]$" for domene som vi vil beregne løsningen på. Dette er initielt satt til $[0, 2] \times [0, 2]$. Dessuten sendes "partition=$[x, y]$", som angir hvor fint gitter vi skal arbeide med. Dette er initielt satt til $[15, 15]$. Dette er verdier som vi gjerne vil ha tilgang til i scriptet, for å sende dem videre til likningsløseren. Som tidligere nevnt, blir disse verdiene lagret i "QUERY_STRING," og må hentes ut fra denne. I midlertid har Morten Roland, på "drift," laget et perl–program som gjør jobben vi har behov for. Dette programmet heter `unpackvar` og ligger i filkatalogen `/SI/WWW/http/etc/` på midgard. Etter å ha kjørt dette programmet, vil det som ligger i "QUERY_STRING" bli lagt i variable "FORMVAR_navn=verdi," for hvert "navn=verdi" par.

I tillegg til at `unpackvar` dekoder "QUERY_STRING" er det mulig å gjøre noe feilsjekking med programmet, for å se om det er et lovlig kall til cgi–script som kommer fra klienten. Vårt lille eksempel vil da se slik ut:

```
#!/bin/sh
eval "‘/SI/WWW/http/etc/unpackvar -export-sh‘"
if test -n "$FORMERROR"; then
   /SI/WWW/http/etc/unpackvar -report "$FORMERROR"
   exit 0
fi

app < input.i
simres2mtv -f SIMULATION -s -n u -a
#START OUTPUT OF DATA
echo "200 OK"
echo "Content-type: text/plain"
echo ""
cat SIMULATION.scalar.mtv
```

Vi ser her også demonstrert et annet viktig punkt, nemlig at vi kaller programmer med filkatalogen spesifisert helt fra "/". Grunnen til dette er at det ikke alltid er lett å avgjøre hvor cgi–scriptet kjøres i fra, siden dette kalles fra http. Det er derfor en god vane å spesifisere script og programmer som kalles, med full filkatalog. Dette må altså gjøres både med applikasjonen og med `simres2mtv`.

For å få sendt parameterne fra "GraphView" inn i applikasjonen, må vi modifisere input–data'ene. Det er derfor enklest, i hvertfall i de tilfellene hvor `input.i` bare består av en til to linjer, å sende alle data eksplisitt fra scriptet. Vi ser her et eksempel på hvordan det kan gjøres, der vi også har tatt med eksempler på hvordan "FORMVAR"– variablene kan benyttes.

```
echo "d=2 $FORMVAR_domain
n d=2, elm_tp=ElmB4n2D  div=$FORMVAR_partition
grading=[1,1]" | /SI/WWW/http/scripts/diffpack/app
```

Før vi har et script som gir datasett brukbart for GraphView, må vi gjøre en liten justering. Etter at vi har kjørt `simres2mtv` har vi en datafil på riktig format. Denne kan imidlertid inneholde data på formen A.BCDe-n, og dette er et format som "GraphView" ikke forstår[2]. Vi må derfor gjøre litt enkel prossesering på dataene, for eksempel slik:

```
awk '{if( $1 ~ /#/ || $1 ~ /%/ || $1 ~/$/) print; else
{for(i=1;i<=NF;i++)printf("%f ",$i);
printf(" \ n");}}' < /SI/WWW/http/scripts/diffpack/
       SIMULATION.$domain.$partition
```

3.3 Litt fin-tuning av CGI–scriptet

Det er to ting vi ikke har tatt særlig hensyn til i de som er sagt til nå. Det første er hastighet og det andre er muligheten for at flere aksesserer cgi–scriptet samtidig. Det første spørsmålet er det lite å gjøre med, fordi vi vil være begrenset av tiden det tar å kjøre applikasjonen. Det er imidlertid en ting vi kan gjøre, som er gjort i eksempel–scriptet, som allerede er lagt opp, i skrivende stund befinner dette seg på mitt eget brukerområde på den eksterne serveren, på filen `Graphview.cgi.sh` i katalogen `/SI/ekstern/users/aod/Scripts`. Det vi kan gjøre, er å lage ferdig en del datasett for "vanlige" valg av parametere. Først og fremst vil dette gjelde datasettet for default–verdiene av de to parameterne, domain$= [0,2] \times [0,2]$ og partition$= [15,15]$. Det gjelder da bare å velge et hensiktsmessig format for navning av filene som inneholder datasettene, slik at vi lett kan sjekke om datasettet eksisterer, og i så fall sende dette til klienten. Jeg har valgt å gjøre det på denne måten:

- Oversett $[a,b] \times [c,d]$ til $a - b\_c - d$ og $[x,y]$ til $x - y$.
- Sjekk deretter om filen `SIMULATION.`$a - b\_c - d$`.`$x - y$ finnes.
- Hvis filen finnes, send data, ellers lag nytt datasett, og send dette.

Vi har dermed en enkel måte å finne ut om vi har et datasett klart. Dette vil selvfølgelig øke hastigheten en god del for "vanlige" tilfeller. Spørsmålet blir hvor mange datasett som skal være klare, og i såfall hvilke, men det overlater jeg til andre å bestemme. En mulighet kan være å logge hvilke parametere som brukes, og deretter legge til i listen av ferdige datasett.

[2] Dette er noe som kanskje bør rettes på, hvis noen arbeider videre med programmet

Rent teknisk benyttes *sed* for å oversette parameterne, deretter er det vanlig *if*, med bruk av *test -f* `filnavn` for å sjekke om filen finnes. Ellers går det hele som før.

Det andre punktet var muligheten for at to klienter skal aksessere scriptet samtidig. Slik scriptet foreligger, vil vi da få to forskjellige prossesser, som begge kjører samme script, i samme filområde. De vil derfor begge to produsere en fil `SIMULATION.scalar.no` i samme område, i det øyeblikk begge brukerne vil ha et datasett som det ikke finnes noen ferdig–utgave av. Deretter vil begge prossessene prøve å returnere filen. Da kan selvfølgelig hva som helst skje, og det gjelder å være senest ute, siden siste utgave blir liggende. Dette ordnes enkelt ved at vi i scriptet oppretter et unikt filområde for prossessen i maskinens tmp–katalog. Denne katalogen gir vi navnet `/tmp/prossessID/`, hvor prossessID er et tall som vi har tilgang til i scriptet gjennom variabelen $$. Deretter lar vi scriptet skrive data til denne katalogen, sende datasettet til klienten, og så slette katalogen med dets innhold. Dermed er saken i orden, siden prossess–nummeret er unikt.

4 GraphView Kompilering, Interaksjon og Oppbyggning – Noen Hovedtrekk

Vi vil nå gå litt nærmere inn på oppbyggningen av GraphView, mulighet for interaksjon med brukeren, og hva som kanskje bør gjøres med programmet, for at det skal ha større nytte–verdi. I den forbindelse blir det nødvendig å komme nærmere inn på de begrensningene som nå foreligger. Men la oss først se litt på Java generelt.

4.1 Java – kompilering og dokumentasjon

GraphView er et program som er skrevet i det nye, revolusjonerende (hmm) språket Java. Siden dette er et språk som de færreste foreløpig har vært borti, tar vi med noen generelle ord om bruken av Java.

For å kunne kompilere et Java program må man ha en Java–kompilator. Dette er foreløpig ikke standard program vare på SINTEF, slik at man må ut på surfetur for å hente seg en versjon. Det er Sun som har lansert programmeringsspråket, så det er naturlig å begynne å lete der etter kompilatorer, se for eksempel adressen `http://java.sun.com/products/index.html`. Her vil det finnes pekere til kompilatorer. Installasjon av dette greier man nok selv, med en dokumentasjonen som følger med. Etter at kompilatoren er ordentlig installert, er det ganske greit å fortsette. Man bruker kommandoen *javac* `filnavn.java` til å kompilere. Dette

vil resultere i noen nye filer med etternavnet "class", en for hver klasse som er definert i programmet. Dette er viktig å merke seg hvis man vil flytte applikasjonen. Ved C/C++ kompilering, ender man vanligvis med en fil som inneholder applikasjonen, slik at applikasjonen fungerer like fint om man flytter denne ene fila, så fremt man ikke endrer arkitektur. Skal Java applikasjoner fungere etter å ha vært flyttet på, må man passe på å flytte alt man trenger! Dette er blitt et alvorlig punkt for meg etter diverse problemer.

En liten merknad i forbindelse med innstalleringen av java–kompilator på Sun, som er den muligheten man får av Sun (man kan nok søke andre steder etter kompilatorer for andre arkitekturer), er å passe på system–kallet *uname* som benyttes av java–kompilatoren. På enkelte systemer, blant annet her på SINTEF, vil PATH være satt opp slik at man automatisk får en gnu–versjon av *uname*, for eksempel /usr/local/gnu/bin/uname. Denne kan imidlertid ikke brukes (i skrivende stund), slik at man kan bli nødt til å linke direkte til versjonen som finnes i `/usr/bin`, det vil si versjonen som følger med operativ–systemet fra Sun.

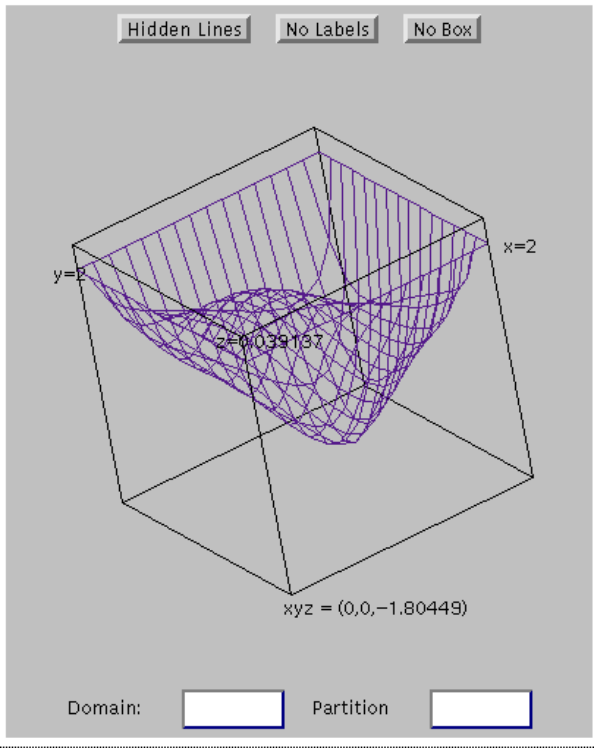
Hvis man ønsker å lære seg litt java, og har bakgrunn i C/C++ verden, slik mange har, kan vi gi noen pekere mot informasjon og dokumentasjon som kan være nyttig. Det finnes en hel del "tutorials" og liknende ute på nettet. Et startpunkt på norsk kan være Morten Lodes kurs, [Lod], dette er en enkel innføring som jeg ikke har sett så mye på selv. . .. Et annet utgangspunkt er å starte enten hos Sun [Sun] eller hos datterselskapet JavaSoft [Jav]. Dette er å gå til selve opphavet, siden Java er utviklet her. En annen peker er å søke etter "java tutorial" på søkeverktøyet HotBot [Hot], og velge fra listen man får opp (som sikkert blir noen tusen). Den beste resurssen for den som vil lære seg språket skikkelig, er å lese i en bok om Java, for eksempel boken *Java in a Nutshell*, [Fla96] som avdeling 4240 har kjøpt, og som kanskje kan finnes ved henvendelse til Gimse eller Ødegård.

Det finnes også endel informasjon om CGI rundt omkring, et startpunkt her for de som ønsker å sette seg inn i det som trengs for å løse oppgavene knyttet til denne pakken, er informasjonen hos NCSA [NCS].

Vi benytter Diffpack applikasjoner for å produsere dataene som vi ønsker å visualisere. I den forbindelse ønsker vi data på format lik det som benyttes av plotte–programmet PlotMtv. Informasjon om disse tingene kan finnes i flere Diffpack–rapporter, et utgangspunkt for denne informasjonen kan finnes i Langtangens rapport om visualisering av skalar– og vektor felt i Diffpack, [Lan96].

4.2 Programmet – oppbyggning

Til slutt vil jeg gå nærmere inn på hvordan Java–programmet "GraphView" er bygd opp, for å gjøre dokumentasjonen mer fullstendig. Dette blir informasjon beregnet på de som eventuelt skal oppgradere programmet til å takle andre, vanskeligere situasjoner enn det er beregnet til idag.



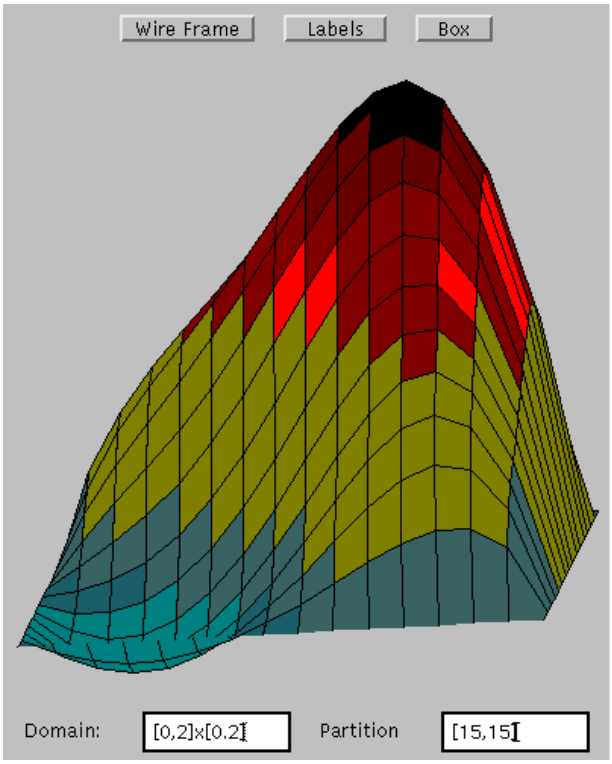
Figur 2: Appleten slik den ser ut i min Netscape!

Som et utgangspunkt kan vi ha det ferdige resultatet som vil komme frem i WWW–programmet, når brukeren går til en side som inneholder applet'en. Bilde i figur 2 vil da møte brukeren. Vi viser også applet'en slik den vil se ut etter at brukeren har trykket litt på knapper og rotert figuren, se figur 3. Her har vi også fylt ut data på lovlig form i tekstboksene i applet'ens nedre kant.

Programmet GraphView er bygd opp omkring tre klasser. Disse har navnen *Matrix3D*, *Model3D* og *GraphView*. Den første av disse har jeg kopiert fra en Java–applet vi fant i Suns distribusjon av Java–kompilatoren, der den var med som et eksempel. Applet'en het der "ThreeD.class". Dette er en matrise–klasse som tar seg av de nødvendige beregningene for rotasjon, translasjon og skalering av et objekt. Klassen inneholder alle de operasjonene en har behov for i denne sammenheng.

Selve kjernen i programmet er klassen *Model3D*. Det er denne klassen som tar seg av innlesning og lagring av dataene om modelen, og klassen har nødvendige metoder for tegning av figuren. Dette er også den klassen som har gjennomgått de største forandringene etter at jeg begynte å arbeide med denne program–pakken.

Den siste klassen, *GraphView*, tar seg av behandlingen av brukerens ønsker.



Figur 3: Appleten med angitte innputdata og fylte polynomer

Det vil si at klassen, ved opptart, initialiserer appleten, med knapper og bokser til å skrive i. Deretter akseseres CGI–scriptet får å få tilgang til nødvendige data. Etter dette opprettes et *Model3D*–objekt, som leser inn data. Etter dette vil *GraphView*–klassen se etter mus–bevegelse, input i tekst–boksene og trykk på knappene for plotting av gitter–figur eller fylte polynomer, akser eller en boks rundt figuren. Dette kalles "events," og behandles av *GraphView*.

Å komme inn på alle programmets enkelheter vil ta for lang tid, jeg vil heller anbefale den som eventuelt skal utvide programmet, om å sette seg grundig inn i kilde koden, den burde være forholdsvis grei å lese. Noen hovedpunkter kan det imidlertid være greit å nevne.

Innlesning av data er basert på det aller enkleste formatet vi kan ha for *plot-MTV*. Vi antar at vi har jevn oppdeling av gitteret , slik at bare funksjons–verdiene er angitt i data settet, og disse er listet opp fortløpende. Dette gjør det enkelt å lese inn data, men svært lite fleksibelt. En første utvidelse bør nok være å utvide programmet med mulighet for å lese inn alle koordinatene for hvert enkelt punkt. Datastrukturen som finnes tar allerede nå vare på alle koordinatene, så dette blir bare en liten endring av innlesnings–delen i initialiserings–prosedyren i *Model3D*.

Hvis man skal endre innlesnings algoritmen, som er plassert i konstruktør–prosedyren i *Model3D*, bør man være klar over at jeg her benytter en klasse fra standard–bibliotekene i Java som heter *Streamtokenizer*. Dette er en klasse som gir en mulighet til å lese en inn-strøm (*Instream*) på en mer funksjonell måte enn tegn for tegn. *Streamtokenizer* vil lese tegn som logisk hører sammen, og returnere dette som et objekt med noen ekstra attributter, deriblant opplysning om det er tekst eller tall som er lest. Klassen ignorer "hvite" tegn og alt etter et kommentar tegn, som man kan velge selv.

Etter at alle data er lest inn, bygges data–strukturen opp ved at punktene i modellen legges inn som elementer i den 3D modellen vi skal plotte. Det som gjøres er å legge alle koordinatene for alle punkter inn i et eget array, slik at vi får mulighet til å transformere figuren slik vi ønsker, ved å operere på et element. Deretter spesifiseres det hvilke punkter vi skal tegne linjer mellom, og hvilke punkter som skal danne polygoner som vi ønsker å fylle. Alt dette er basert på at vi har rektangulære gitter, men kan antagelig brukes så lenge man har sytematiske gitter. Hvis man skal utvide programmet til også å ta hånd om mer rufsete gitter, må man tenke på andre datastrukturer, en første tanke bør kanskje være å definere en polygon–klasse, hvor man kan legge inn koordinater fritt.

Den neste prosedyren hvor spesielle vanskeligheter oppsto, er prosedyren for å plotte modellen. Så lenge vi skal tegne en gitter–figur av de linjene som er spesifisert, uten å ta hensyn til at noen linjer dekker andre, det vil si ligger foran i bildet, er det greit. Men i det øyeblikk vi ønsker å tegne fylte ruter i steden for et gitter, ønsker vi også å skjule det som er bak, det vil si at rekkefølgen vi plotter rutene i, ikke lenger er likegyldig. Jeg har ikke funnet noen fullgod algoritme for å gjøre dette korrekt, men jeg har funnet en som fungerer i de fleste tilfeller. Problemet oppstår når man dreier figuren, spesielt er det vanskelig å finne en algoritme som fungerer godt når figuren dreies slik at z–aksen blir liggende horisontalt i bildet. Plottingen av fylte polygoner foregår i prosedyren *PolygonFill*.

For å kunne plotte figuren riktig, forsøker vi å bestemme hvilket hjørne som til enhver tid er "bakerst", og deretter plotte figuren relativt til dette. Dette bestemmes ved å bruke figurens orientering i koordinatsystemet, og ved å bestemme hvilket hjørne som ligger lengst til venstre. Problemet er at dette ikke er riktig fremgangsmåte når "z–aksen" dreies til å være nesten horisontal. Imidlertid gir dette synsvinkler som man sjelden ønsker å benytte, i motsetning til de vinklene hvor algoritmen fungerer.

Klassen *GraphView* vil jeg ikke si så mye om, den er stort sett selvforklarende når man har satt seg litt inn i Java–funksjonene. Etter at modellen er initialisert består *GraphView* stort sett bare av prosedyrer for å registrere og behandle brukerens interaksjon. Dette igjen består av transformasjoner av modellen ved å bruke objekter av matrise klassen *Matrix3D* og funksjonene i denne.

Referanser

[Fla96] Daviv Flangan, *Java in a Nutshell,* A Nutshell Handbook, O'Reilly and Associates, Inc, 1996.

[Hot] *Søke–verktøyet HotBot*, http://www.hotbot.com/.

[Jav] *Hjemmeside for JavaSoft, datter–selskap av Sun, spesialisert på java*, http:// www.javasoft.com.

[Lan96] Hans Petter Langtangen, *An Example on Visualizing of Scalar and Vector Fields in Diffpack*, Tech. report, SINTEF, UiO, 1996.

[Lod] Morten Lode, *Javakurs,* Publisert på internett på adressen http://www.grm.hia.no/~morlo/javakurs.

[NCS] *CGI–dokumentasjon hos NCSA,(National Center for Supercomputing Applications)*, http://hoohoo.ncsa.uiuc.edu/cgi/overview.html.

[Sun] *Hjemmesiden til Sun*, http://www.sun.com/, Java–side på adressen http://www.sun.com/960601/index.java.html.

List of STIM Working Notes

stimwn0000: *How to Write STIM Working Notes*
(by Are Magnus Bruaset)

stimwn0001: *Efficient Memory Handling in C++ Programs*
(by Erlend Arge)

stimwn0002: *Some data structures in Siscat*
(by Kyrre Strøm)

stimwn0003: *Introductory Exercises to the Siscat Library*
(by John Kofi Arthur)

stimwn0004: *An Appetizer on the BasicTools Menu*
(by Geir Westgaard)

stimwn0005: *Handling handles*
(by Johannes Kaasa)

stimwn0006: *Project Management Tools User Guide*
(by John Kofi Arthur)

stimwn0007: *Visualizing Diffpack Computations in Matlab*
(by Johan Berge)

stimwn0008: *A set of generic makefiles*
(by Trond Vidar Stensby)

stimwn0009: *GraphView - En Java og CGI-pakke for 3D Grafikk*
(by Åsmund Ødegård, Gerhard W. Zumbusch)