Språk

De följande sidorna - som inte kommer att gås igenom - diskuterar några programspråk och miljöer som kan användas tillsammans med OpenGL. Syftet är framför allt att tala om vad som behöver göras för att ett visst språk skall kunna användas i vårt Linux-system. Därmed vänder sidorna sig mest till den som redan kan och i datorgrafikkursen vill begagna ett "annorlunda" språk. Syftet är inte att lära ut ett nytt språk. Trots det ger jag något exempel och några kommentarer.

Det vanligaste programspråket vid programmering mot OpenGL är sannolikt C++. För enklare tillämpningar duger det mindre komplicerade C väl. Men vi kommer åt OpenGL från en mängd olika språk.

Datorgrafiska program är lämpade för objektorienterade programmering, vilket betyder t ex C++ eller Java. Många av deltagarna i denna kurs brukar sakna kunskaper i C eller C++, men vill passa på att lära sig något av dessa språk.

Man skulle kunna ta upp flera andra språk, t ex Haskell, men detta urval får räcka.

Språk: C++ 1(4)

Eftersom de flesta numera behärskar Java, kanske jag skulle försöka leda in några fler på C++-spåret (man klarar sig dock med C eller Java!). En fördel med detta (precis som med Java) skulle vara att det är enklare att göra program med bra användargränsnitt. C++ har inte egna bibliotek för det, men det finns flera fria sådana skrivna i C++. Vi har nämnt FLTK (`http://www.fltk.org`) och borde nämnt FOX (`http://www.fox-toolkit.org`).
Det är lätt att från C++ nå programvara skriven i C. Detta förklarar att GLUT och OpenGL kan anropas precis som från C. Däremot kan man i allmänhet inte anropa C++-bibliotek från C. Detta är orsaken till att FLTK, FOX och för den delen DirectX inte gärna används från C-program.
Jag vill ge en liten introduktion till C++ med betoning på klassbegreppet. För att kunna arbeta seriöst med C++ krävs mycket mer; det är långtifrån så enkelt som man kan tro av mitt exempel. Du hänvisas till annan litteratur (t ex fri på nätet) för fler detaljer.

Exempel: Vi gör ett program som arbetar med punkter i 2D med heltalskoordinater. Vi inför därför en klass *Point* för sådana punkter. Programmet `1.cpp` kompileras och körs så här:

```
>g++ 1.cpp -o 1.out
>1.out
90
120
```


Språk: C++ 2(4)

Programmet följer

```
>cat 1.cpp
#include <iostream>
// Specifikation
class Point {
     private:
          int x,y;
     public:
          Point(int x = 0, int y = 0);
          int getX();
          void setX(int x);
};
// Implementeringen
Point:: Point(int x, int y) {
         this->x = x; this->y = y;
     }
int Point::getX() { return x;}
void Point::setX(int x) {
     this->x = x;
}
// Resten av programmet
int main(int argc, char* argv[]) {
     Point a(10,20), b(30,40);
     Point* c = new Point(50,60);
     a.setX(90); c->setX(120);
     cout << a.getX() <<'\n';
     cout << c->getX() <<'\n';
}
```


Språk: C++ 3(4)

Exekveringen startar alltid i `main`. Där skapar vi punkterna *a* och *b* som vanliga variabler och punkten *c* som en dynamisk variabel. I det senare fallet används explicit pekarnotation, medan vi i det första använder samma notation som i Java. Lagringsmässigt är däremot Javas sätt mera likt det andra sättet. Utskrifter görs med s k strömteknik (man kan även använda C:s `printf`). Specifikationen för klassen Point deklareras i klassdeklarationen, som delvis liknar ett gränssnitt i Java. Implementeringen (av konstruktor och metoder) görs utanför klassen. Orden **`private`** och **`public`** har samma syfte som i Java, men används tydligen litet annorlunda.

Något mera proffsigt är att lägga klassen med sin implementering i två separata filer. En för specifikationen och en för implementeringen. Programmet består nu av de tre filerna `2.cpp`, `Point.h` och `Point.cpp`.

```
>cat 2.cpp
#include <iostream>
#include "Point.h"
int main(int argc, char* argv[]) {
     Point a(10,20), b(30,40);
     Point* c = new Point(50,60);
     a.setX(90); c->setX(120);
     cout << a.getX() <<'\n';
     cout << c->getX() <<'\n';
}
```

Språk: C++ 4(4)

```
>cat Point.h
class Point {
     private:
          int x,y;
     public:
          Point(int x = 0, int y = 0);
          int getX();
          void setX(int x);
};
```

```
>cat Point.cpp
#include "Point.h"
Point:: Point(int x, int y) {
         this->x = x; this->y = y;
}
int Point::getX() {
     return x;
}
void Point::setX(int x) {
     this->x = x;
}
```

Kompilering, länkning och körning får nu göras enligt (man har inte samma automatik som i Java; en s k makefil skulle underlätta)

```
>g++ -c Point.cpp
>g++ 2.cpp Point.o -o 2.out
>2.out
90
120
```

Anrop av C-kod från C++ kan kräva extern “C” { ... }. Se t ex inkluderingsfilen `glut.h`.

Språk: C# 1(3)

Språket C# är Microsofts svar på Java från Sun. Tillkom åtminstone delvis till följd av ett bråk mellan bolagen. Huvudansvarig var Anders Hejlsberg (dansk liksom Bjarne Stroustrup, som gjorde C++). Han sa vid något tillfälle: “*First of all, C# is not a Java clone. In the design of C#, we looked at a lot of languages. We looked at C++, we looked at Java, at Modula 2, C, and we looked at Smalltalk. There are just so many languages that have the same core ideas that we’re interested in, such as deep object-orientation, object-simplification, and so on.*”

C#-program arbetar man normalt med bara på Windows-plattformen. Men det finns ett system Mono, vars syfte är “*an effort to implement the .NET Framework to Unix*”, som gör det möjligt att använda språket även under Linux. För att koppla ihop C# och OpenGL kan man använda ett annat system Tao, som finns för både Windows och Linux (jag har installerat det bara för Linux). Liksom alla projekt av den här typen är Mono och Tao beroende av ideella krafter, som när som helst kan ta slut. Tao stöder förmodligen inte hela OpenGL 2.0.

C# är likt Java, men avviker i vissa avseenden. Själva språket går vi inte på här, utan vi koncentrerar oss på OpenGL-kopplingen, som är enklare än den för Java. Man kan också anropa C från C# (litet enklare än i Java).

C# och OpenGL under Linux sker så här:
Först: `source /users/course/TDA360/LINUX/setupMono`
Kompilera CS.cs till CS.exe med:

```
mcs  CS.cs -r:Tao.FreeGlut.dll -r:Tao.OpenGl.dll
```

Kör ett program CS.exe med: `mono CS.exe`

Språk: C# 2(3)

I `/users/course/TDA360/LINUX/MONO/TAO/` finns flera körbara program, t ex `Redbook.Cube.exe` eller det häftigare `NateRobins.Maiden.exe`. I princip skall väl `.exe`-filerna gå att köra under Windows, men jag råkar ut för Permissions-problem.

Här följer ett exempel GL_3D_2003.c från tidigare OH i C#-form.

```
using System;
using Tao.FreeGlut;
using Tao.OpenGl;
public sealed class GL_3D_2003 {
    // Här startar körningen
    public static void Main(string[] args) {
        Glut.glutInit(); // Brukar inte behövas i C
        // Dubbla bildminnen vid all rörelse
        Glut.glutInitDisplayMode(Glut.GLUT_DOUBLE | Glut.GLUT_DEPTH
                                 | Glut.GLUT_RGB);
        Glut.glutInitWindowSize(400,400);
        Glut.glutCreateWindow("Snurrande triangel: objektbyte=b,
                               punktbyte=c");
        Glut.glutReshapeFunc (new Glut.ReshapeCallback(storleksbyte));
        Glut.glutDisplayFunc(new Glut.DisplayCallback(rita));
        Glut.glutIdleFunc(new Glut.IdleCallback(inget_att_gora));
        Glut.glutKeyboardFunc(
               new Glut.KeyboardCallback(tangenthanterare));
        Gl.glEnable(Gl.GL_DEPTH_TEST);
        Glut.glutCreateMenu(new Glut.CreateMenuCallback(myMenu));
        Glut.glutAddMenuEntry("Avsluta",9);
        Glut.glutAttachMenu(Glut.GLUT_RIGHT_BUTTON);
        Glut.glutMainLoop();
    }
    private static double vinkel = 0.0;
    private static bool POS1 = true, OBJ1 = true;
    private static void rita() {
        Gl.glClearColor(1.0f,1.0f,1.0f,0.0f);
        Gl.glClear(Gl.GL_COLOR_BUFFER_BIT
                 | Gl.GL_DEPTH_BUFFER_BIT);
        Gl.glMatrixMode(Gl.GL_MODELVIEW);
        Gl.glLoadIdentity();
        if (POS1) Glu.gluLookAt(0,0,2, 0,0,0, 0,1,0);
        else Glu.gluLookAt(1,1,1, 0,0,0, 0,1,0);
```


Språk: C# 3(3)

```
        koordinataxlar(2.0);
        Gl.glRotated(vinkel,0,1,0);
        if (OBJ1) {
            Gl.glBegin(Gl.GL_POLYGON);
                Gl.glColor3f(1,0,0); Gl.glVertex3d(0,0,0);
                Gl.glColor3f(0,1,0); Gl.glVertex3d(2,0,0);
                Gl.glColor3f(0,0,1); Gl.glVertex3d(2,1,0);
            Gl.glEnd();
        } else {
            Gl.glColor3d(1,0.5,0);
            Glut.glutSolidTeapot(1.0);
        }
        Glut.glutSwapBuffers();
    }
    private static void tangenthanterare(byte key, int x, int y) {
        if (key == 'c') POS1 = !POS1;
        if (key == 'b') OBJ1 = !OBJ1;
        if (key == 'q') Environment.Exit(0);
        Glut.glutPostRedisplay();
    }
    private static void myMenu(int item) {
        switch(item) {
            case 9: Environment.Exit(0); break;
        }
    }
    private static void inget_att_gora() {
        vinkel = vinkel + 0.1;
        Glut.glutPostRedisplay();
    }
    private static void storleksbyte(int width, int height) {
        Console.WriteLine("Ändrad fönsterstorlek!");
        Gl.glViewport(0, 0, width, height);
        Gl.glMatrixMode(Gl.GL_PROJECTION);
        Gl.glLoadIdentity();
        Glu.gluPerspective(120,1,0.1,10);
    }
    private static void koordinataxlar(double L) {
        Gl.glColor3f(0,0,0); // Svart
        Gl.glBegin(Gl.GL_LINES);
           Gl.glVertex3d(-L,0,0); Gl.glVertex3d(L,0,0); //x-axel
           Gl.glVertex3d(0, -L,0); Gl.glVertex3d(0, L,0); //y-axel
           Gl.glVertex3d(0, 0,-L); Gl.glVertex3d(0, 0,L); //z-axel
        Gl.glEnd();
    }
}
```

Språk: Python 1(3)

Python[1] är ett modernt **interpreterande** språk, som har en del föregångare. Det har olika former av objekt-orientering och kan knytas till OpenGL (kopplingen stöder inte OpenGL 2.0 fullt ut) och GLUT. På ytan är det likt både C och Matlab. En trevlig sak är att man kan arbeta med komplexa tal lika enkelt som med vanliga. När det gäller enklare OpenGL-program är skillnaden mot C i stort sett bara syntaktisk.

Du kan i vår Linux-miljö f n köra Python (inkl OpenGL) genom att först skriva

```
source /users/course/TDA360/LINUX/bin/setupPYOGL
```

och sedan (om du vill få en färskare version än vad `python` ger)

```
pythonMB
```

Python kan användas interaktivt, t ex

```
>pythonMB
Python 2.4.1 (#1, Jun 10 2005, 16:11:44)
[GCC 3.2.3 20030502 (Red Hat Linux 3.2.3-52)] on linux2
Type "help", "copyright", "credits" or "license" for more
information.

>>> a=7
>>> a*a
49
>>> b=3.14
```

1. Det är faktiskt uppkallat efter BBC-serien “Monty Python’s Flying Circus”.

Språk: Python 2(3)

```
>>> b
3.1400000000000001
>>> c=2+5.7j
>>> c+c
(4+11.4j)
>>> for x in [4,7,8]:
...    print x
...
4
7
8
>>>
```

eller för den delen

```
>>> from OpenGL.GL import *
>>> from OpenGL.GLUT import *
>>> glutCreateWindow("Hej")
1
>>> def display():
...     return
...
>>> def reshape(w,h):
...     return
...
>>> glutDisplayFunc(display)
>>> glutReshapeFunc(reshape)
>>> glutMainLoop()
```

varefter ett fönster visar sig.

Man avslutar med CTRL-D. För vår del är det dock naturligast att arbeta mot filer. T ex

`pythonMB GL_3D_2003.py`

kör det program som vi tidigare skrivit i C och C#.

Språk: Python 3(3)

Programmet `GL_3D_2003.py` ser ut som följer:

```
#!
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
import sys


# Globala variabler
vinkel = 0.0
POS1 = 1; OBJ1 = 1

def koordinataxlar(L):
       glColor3f(0,0,0)
       glBegin(GL_LINES)
       # Python gillar inte ytterligare indrag
       glVertex3f(-L,0.0,0.0); glVertex3f(L,0,0); #x-axel
       glVertex3f(0, -L,0); glVertex3f(0, L,0); #y-axel
       glVertex3f(0, 0,-L); glVertex3f(0, 0,L); #z-axel
       glEnd()

def rita():
    global vinkel, POS1, OBJ1
    glClearColor(1.0,1.0,1.0,0.0)
    glClear(GL_COLOR_BUFFER_BIT
                  | GL_DEPTH_BUFFER_BIT)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    if (POS1):
        gluLookAt(0,0,2, 0,0,0, 0,1,0)
    else:
        gluLookAt(1,1,1, 0,0,0, 0,1,0)

    glPushMatrix()
    koordinataxlar(2.0)
    glRotated(vinkel,0,1,0)
```



```
    if (OBJ1):
      glBegin(GL_POLYGON)
      glColor3f(1,0,0); glVertex3f(0,0,0);
      glColor3f(0,1,0); glVertex3f(2,0,0);
      glColor3f(0,0,1); glVertex3f(2,1,0);
      glEnd()

    else:
      glColor3f(1,0.5,0);
      glutSolidTeapot(1.0);

    glPopMatrix()
    glutSwapBuffers()

def storleksbyte(width, height):
    glViewport(0, 0, width, height)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(120,1,0.1,10)

# args[0] star for key
def tangenthanterare(*args):
   global POS1, OBJ1
   if (args[0] == 'c'):
       POS1 = not POS1

   if (args[0] == 'b'):
        OBJ1 = not OBJ1

   if (args[0] == 'q'):
        sys.exit(0)

   # 27 duger inte p g a bug. Oktalt i stallet.
   if (args[0] == 27 or args[0] == '\033'):
        sys.exit(0)

   glutPostRedisplay()
```

```
def myMenu(item):
    sys.exit(0)

def inget_att_gora():
    global vinkel
    vinkel = vinkel + 0.1
    glutPostRedisplay()

def menu(item):
    sys.exit(0);

def main():
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH
                                          | GLUT_RGB);
    glutInitWindowSize(500,400);
    glutCreateWindow("Snurrande triangel: objektbyte=b,
                                  punktbyte=c. Meny MK3");
    glutReshapeFunc (storleksbyte);
    glutDisplayFunc(rita);
    glutIdleFunc(inget_att_gora);
    glutKeyboardFunc(tangenthanterare);
    glEnable(GL_DEPTH_TEST);
    glutCreateMenu(myMenu);
    glutAddMenuEntry("Avsluta",9);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutMainLoop();

# Skriv meddelande och starta main
print "ESC eller MK3/Avsluta bryter programmet"
main()
```

Några observationer kring Python.

- **Globala variabler**: Om en sådan skall användas i en funktion, måste den deklareras i funktionen som global variabelnamnet.
- **Variabler typas ej explicit**:
- **Sammansatta satser** (if, for, while): I stället för avgränsare som { och } i C och Java används : och tom rad. De ingående satserna måste dras in!

- **Funktionsdefinition**: Inleds med def funktion(parameternamn): och avslutas med tom rad. Satserna i den måste dras in.
- **Lista**: T ex [2,7,3], range(5), range(3,7).
- **for-snurra**: for variabel in lista:

Se ytterligare OpenGL-exempel i t ex `/users/course/TDA360/LINUX/PYTHON/PyOpenGL-2.0.1.09/OpenGL/Demo/twburton` och dito `.../NEHE`