Előkészületek

Ebben a rövíd leírásban néhány ötletet mutatunk be a dinamikus rendszerek 2-es számú projektlehetőségének megvalósításával kapcsolatban. Leginkább a grafikus felülettel, alakzatok kirajzolásával foglalkoztunk az eddigi munkánk során.

Melyek a céljaink?

- Apophysis-hez hasonló program kivitelezése, amely segítségével fraktál transzformációkat végezhetünk
- Nyílt forráskódú alkalmazás készítése GNU/Linux operációs rendszer alatt
- Ismerkedés a számítógépes grafikáva
- Matematikai ismeretek szerzése

Milyen eszközöket használnánk?

- Programozási nyelv: C/C++
- Dinamikus könyvtárak felhasználása: OpenGL, Glut, GTK# stb.
- Verziókezelő rendszer: Git
- Fejlesztői környezet: Eclipse

Miért pont ezt a feladatot választottuk?

Szívesen dolgoznánk olyan feladattal amelyben számítógépes grafikával kapcsolatos alapismereteket sajátíthatnánk el, másrészt úgy érezzük, hogy néhány matematikával kapcsolatos fogalom is jobban letisztulna számunkra.

Kik alkotják a csapatot?

A csapat két tagból áll:

- Szász Attila
- Szabó Zoltán

Hol lehet nyomonkövetni a fejlesztést?

A programmal kapcsolatos minden anyagot ide fogunk feltölteni:
https://github.com/szaza/materials

Határidő: 2013 május 31

Eszközök:

1. A **GIT** nevű verziókezelő rendszert fogjuk használni.

   Rövid magyarnyelvű leírás a git használatáról:
   http://wiki.hup.hu/index.php/Hogyan_haszn%C3%A1ljuk_a_Git_verzi%C3%B3kezel%C5%91_rendszert

2. **Eclipse** - http://www.eclipse.org
   CDT fejlesztői környezet http://www.eclipse.org/cdt/ (c++ programozáshoz, tartlamazza a compilert is).

3. **Open Office** – a dokumentáció elkészítéséhez
   http://www.libreoffice.org/

Ismerkedés az OpenGl-el és a Glut-al

Glut telepítése:

- a glut3 vagy freeglut3 csomag megkeresése és telepítése.
- Figyelem: 64 bit-es rendszer esetén ajánlott a freeglut3:i386 telepítése, ez történhet software-managerből, vagy elérhető a *http://freeglut.sourceforge.net/* címen.
- Szükséges még telepíteni a freeglut3-dev nevű csomagot. Ezt a következő parancs futattásával tehetjük meg *sudo aptitude install freeglut3-dev*

OpenGl programok fordítása:

- A fordítást egy *Makefile* segítségével hajtjuk végre.
- Hozzunk létre egy *Makefile* nevű állományt és írjuk bele a következőket

```
#Makefile
CC = g++
APPS = helloworld
OBJ = $(APPS).o
SRC = $(APPS).cpp

CFLAGS = $(C_OPTS) -I/usr/include
#Az lxi opció gondot okozhat egyes disztribucióknál
LIBS = -L/usr/X11R6/lib -lX11 -lxi -lglut -lGL -lGLU -lm -lpthread
#Nállam Linux Mint alatt az lxi kapcsoló hibát okozott ezért kivettem

application:$(APPS)

clean:
	rm -f $(APPS) *.raw *.o core a.out

realclean:	clean
	rm -f *~ *.bak *.BAK

.SUFFIXES: cpp o
.cpp.o:
	$(CC) -c $(CFLAGS) $<

$(APPS): $(OBJ)
	$(CC) -o $(APPS) $(CFLAGS) $(OBJ) $(LIBS)

depend:
	makedepend -- $(CFLAGS) $(SRC)
```

Egy egyszerű program

Hozzuk létre az első példaprogramot. Nyissunk meg egy szövegszerkesztőt, majd írjuk bele a következőket és mentsük le helloworld.cpp néven.

```
#include <GL/glut.h>

void display(void) {}

int main(int argc, char **argv)
{
	glutInit(&argc, argv);//initializalja az open gl-t
	glutCreateWindow("Hello Vilag!");//letrehozza az ablakot
	glutDisplayFunc(display);//ha az ablakot ujra kell rajzolni meghivja a display fugvenyt
	glutMainLoop();//elinditja a programszalat
	return 0;
}
```

Ezután futassuk a parancssorból a Makefile-t:
$ *make*

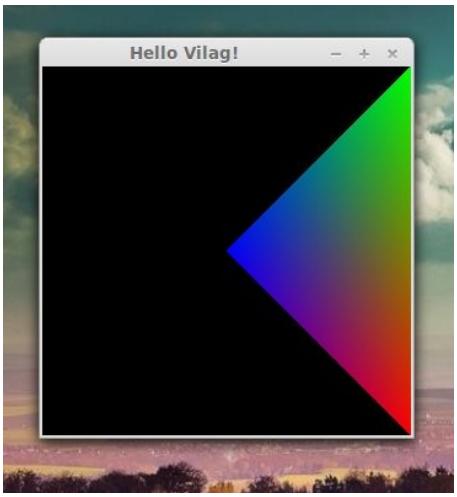
Létrejön a helloworld nevű bináris futtatható állomány és egy helloworld.o kiterjesztésű állomány. Futtassuk az első programunkat:
$ ./helloworld

Rajzolás az ablakba

Addjuk a meglévő helloworld nevű állományhoz a következő függvényt:

```
void display(void)
{
	glClear(GL_COLOR_BUFFER_BIT); //torli a parametereket
	//GL_COLOR_BUFFER_BIT -a kepernyo buffere
	//GL_DEPHT_BUFFER_BIT -a melyseg buffer
	glBegin(GL_TRIANGLES);  //begin end teg, meg kell adni a rajzolni kivant primitivek tipusat pl GL_TRIANGLES, GL_LINES, GL_QUADS
		glColor3f(0.0, 0.0, 1.0);//aktualis szinbeallitas
		glVertex2i(0, 0);//csucspontok koordinataja
		glColor3f(0.0, 1.0, 0.0);
		glVertex2i(1,1);
		glColor3f(1.0, 0.0, 0.0);
		glVertex2i(1, -1);
	glEnd();
	glFlush();//kikenyszeriti a rajzolast
}
```

A következő ábra kell megjelenjen:




```
glColor3f(0.0, 0.0, 1.0); //aktualis szinbeallitas
glVertex2i(0, 0); //csucspontok koordinataja
```

Módosítsuk a csúcspontok koordinátáit, és próbáljuk ki többféleképpen.
Próbáljunk négyszöget rajzolni az alábbi kód segítségével.

```
glBegin(GL_QUADS);
                glColor3f(0.0, 0.0, 1.0);//aktualis szinbeallitas
                glVertex2i(-1,-1);//csucspontok koordinataja
                glColor3f(0.0, 1.0, 0.0);
                glVertex2i(1,-1);
                glColor3f(1.0, 0.0, 0.0);
                glVertex2i(1,1);
                glColor3f(.0, 0.0, 1.0);
                glVertex2i(-1,1);
glEnd();
```

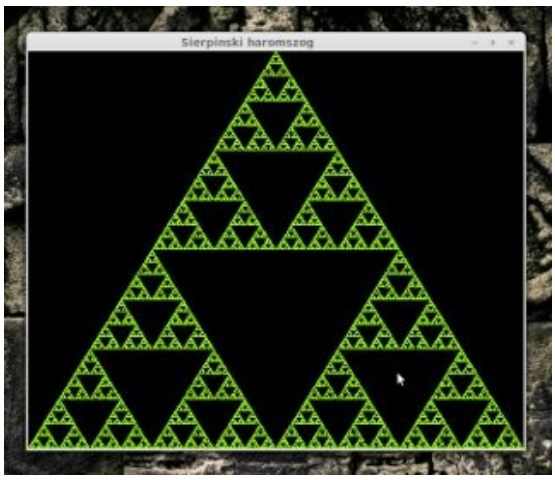
További alakzatok kódjait találhatjuk a http://www.opengl.org/wiki/Primitive címen. (OpenGl primitives)

Kísérlet fraktál kirajzolásra

Sierpinski háromszög

Cél: Próbáljunk meg közismert lehetőleg egyszerű fraktálokat kirajzolni.
Legyen az első próbálkozás a sierpinski háromszög.

Hozzunk létre egy pont típust amibe tárolni fogjuka pont x,y koordinátáit.



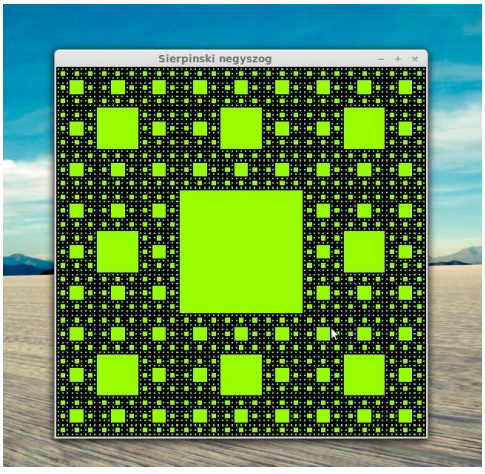

```
//Pont típus létrehozása
struct Pont {
  GLfloat x, y;
  Pont(GLfloat x = 0, GLfloat y = 0): x(x), y(y) {}
  Pont midPont(Pont p) {return Pont((x + p.x) / 2.0, (y + p.y) / 2.0);}
};
```

Állítsunk be néhány alaptulajdonságot: háttérszín, szín, nézőpont.

```
void init() {
  glClearColor(0.0, 0.0, 0.0, 0.0);    //Háttérszín beállítása
  glColor3f(0.6, 1.0, 0.0);            //Az ecset színének a beállítása

  //Az ablak alapbeállításai, a kamera nézőpontjainak a beállításai
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(0.0, 500.0, 0.0, 500.0, 0.0, 1.0);
}

//Kirajzolás meghívása
void display(void) {
        glClear(GL_COLOR_BUFFER_BIT);        //A felület törlése
        //A kezdeti háromszög csúcspontjainak a beállítása (a nagy háromszög)
        static Pont csucs[] = {Pont(0, 0), Pont(250, 500), Pont(500, 0)};
        static Pont p = csucs[0];    //A p pont legyen egyenlő az első ponttal

        glBegin(GL_POINTS);
        for (int k = 0; k < 100000; k++) {
                //Kiszámítja a két pont közötti távolság felét
                p = p.midPont(csucs[rand() % 3]);
                glVertex2f(p.x, p.y);
         }
         glEnd();
         glFlush();
}
```

Sierpinski négyszög

Sierpinski négyszög megvalósítása rekurzió segítségével.




```
#include <GL/glut.h>
#include <iostream>

using namespace std;
```

```
//Pont típus létrehozása
struct Pont {
  GLfloat x, y;
  Pont(GLfloat x = 0, GLfloat y = 0): x(x), y(y) {}
  Pont pontKoord(Pont p) {return Pont((2.0*x+p.x)/3.0,(2.0*y+p.y)/3.0);};
  Pont pontKoord2(Pont p) {return Pont((x+2.0*p.x)/3.0,(y+2.0*p.y)/3.0);};
};
```

OpenGl segítségével, négyszögeket rajzolok ki.

```
void drawRect(Pont p1, Pont p2) {
        glBegin(GL_QUADS);
                glVertex2f(p1.x,p1.y);
                glVertex2f(p2.x,p1.y);
                glVertex2f(p2.x,p2.y);
                glVertex2f(p1.x,p2.y);
         glEnd();
}
```

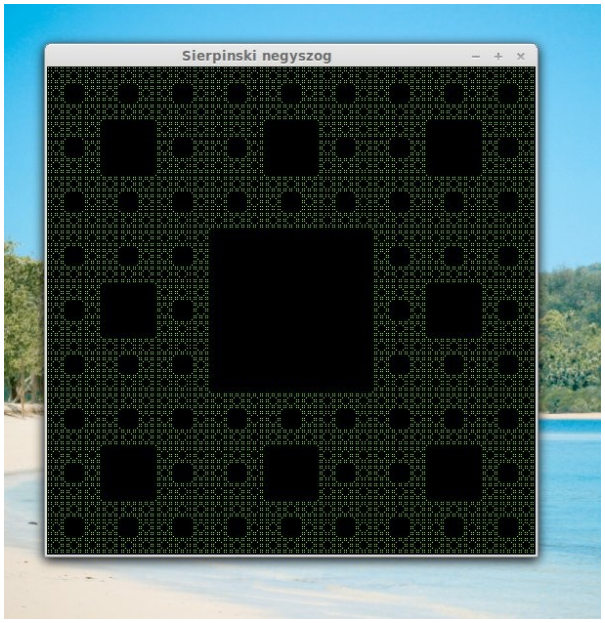
Rekurzív függvény megírása:

```
void drawSquare(int n,Pont p1,Pont p2) {
        drawRect(p1.pontKoord(p2),p1.pontKoord2(p2));

        if (n>0)
        {
                //Bal átlósan lefele
                drawSquare(n - 1, Pont(p1.x, p1.y),Pont((2 * p1.x + p2.x) / 3.0, (2 * p1.y + p2.y) / 3.0));
                //Lefele rajzolja ki
                 drawSquare(n - 1, Pont((2 * p1.x + p2.x) / 3.0, p1.y),Pont((p1.x + 2 * p2.x) / 3.0,(2 * p1.y + p2.y) / 3.0));
                //Jobb átlósan lefele
                drawSquare(n - 1, Pont((p1.x + 2 * p2.x) / 3.0, p1.y),Pont(p2.x,(2 * p1.y + p2.y) / 3.0));
                //A négyzettől balra rajzol
                 drawSquare(n - 1, Pont(p1.x,(2 * p1.y + p2.y) / 3.0),Pont((2 * p1.x + p2.x) / 3.0,(p1.y + 2 * p2.y) / 3.0));
                //A négyzettől jobbra levőt rajzolja ki
                 drawSquare(n - 1, Pont((p1.x + 2 * p2.x) / 3.0, (2 * p1.y + p2.y) / 3.0),Pont(p2.x, (p1.y + 2 * p2.y) / 3.0));
                //A bal átlósan a négyzet fölött rajzol
                drawSquare(n - 1, Pont(p1.x , (p1.y + 2 * p2.y) / 3.0),Pont((2 * p1.x + p2.x) / 3.0,  p2.y));
                //A négyzet fölött rajzol
                drawSquare(n - 1, Pont((2 * p1.x + p2.x) / 3.0, (p1.y + 2 * p2.y) / 3.0),Pont((p1.x + 2 * p2.x) / 3.0,p2.y));
                //Jobb átlósan a négyzet fölött
                drawSquare(n - 1, Pont((p1.x + 2 * p2.x) / 3.0, (p1.y + 2 * p2.y) / 3.0),Pont(p2.x,p2.y));
        }
```

```
}
```

Display függvény megírása:

```
//Kirajzolás meghívása
void display(void) {
        glClear(GL_COLOR_BUFFER_BIT);
        static Pont p = Pont(0,0);
        static Pont p1 = Pont(500,500);

        drawSquare(4,p,p1);

        glFlush();
}
```



Inicializálófüggvény:

```
void init() {
  glClearColor(0.0, 0.0, 0.0, 0.0); //Háttérszín beállítása
  glColor3f(0.6, 1.0, 0.0);                    //Az ecset színének a beállítása

  //Az ablak alapbeállításai, a kamera nézőpontjainak a beállításai
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(0.0, 500.0, 0.0, 500.0, 0.0, 1.0);
}
```

Main függvény:

```
int main(int argc, char **argv)
{
        glutInit(&argc, argv);//initializalja az open gl-t
        glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(40, 40);
        glutCreateWindow("Sierpinski negyszog");//letrehozza az ablakot
        glutDisplayFunc(display);//ha az ablakot ujra kell rajzolni meghivja a display fugvenyt
        init();
        glutMainLoop();//elinditja a programszalat
        return 0;
}
```

A fenti ábrát akkor kapjuk, ha a GL_QUADS primitíveket GL_POINTS-ra cseréljük.