# Laboratorul 2 – Imagine + Imagine + Imagine + ... = Animatie !

**Obiective:**

- Ce este animatia ?
- Hello Animated World !
- Elementele din program ce determina animatia
- Pozitie, Viteza, Acceleratie
- Interactiunea cu tastatura

## Ce este animatia ?

Animatia constituie o iluzie optica a miscarii, creata prin derularea unor imagini reprezentand elemente statice ale acesteia. In productia cinematografica, acest termen se refera la tehnici prin care fiecare secventa a unui film este realizata individual, deci o analiza a miscarii.

Cand secventele sunt derulate una cate una cu o anumita viteza (in cinematografie cu o frecventa de 24 imagini/secunda), deci o sinteza a miscarii, rezulta o iluzie de miscare continuă, prin exploatarea unor factori psihologici ai vederii, memoria asociativa si persistenta retiniana.

## Hello Animated World !

```
#include <GL/freeglut.h>
#include <stdlib.h>

#include <stdio.h>
#include <math.h>

void initGL(int width, int height)
{
    const GLfloat light_ambient[] = { 0.0f, 0.0f, 0.0f, 1.0f };
    const GLfloat light_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
    const GLfloat light_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
    const GLfloat light_position[] = { 2.0f, 5.0f, 5.0f, 0.0f };

    glEnable(GL_LIGHT0);
    glEnable(GL_LIGHTING);
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);

    glEnable(GL_CULL_FACE);
    glCullFace(GL_BACK);

    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LESS);

    glEnable(GL_COLOR_MATERIAL);
```

```
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0f, (GLfloat)width/(GLfloat)height,
    2.0f, 100.0f);
    glMatrixMode(GL_MODELVIEW);
}

static void
display(void)
{
    static int frame,timebase=0;
    int time;
    char s[100];
    frame++;
    time=glutGet(GLUT_ELAPSED_TIME);
    if (time - timebase > 1000)
    {
        sprintf(s,"[FPS:%4.2f]  Lab  2:  Imagine+Imagine+Imagine+...
= Animatie !",frame*1000.0/(time-timebase));
        glutSetWindowTitle(s);
        timebase = time;
        frame = 0;
    }

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glLoadIdentity(); //initializarea sistemului de coordonate

    static float axisRot = 0.0f;
    static float globRotR = 0.0f;
    static float globRotG = 120.0f;
    static float globRotB = 240.0f;

    glColor3f(1.0f, 0.0f, 0.0f);
    glPushMatrix();
        glTranslatef(0.0f,0.0f,-20); //deplasat pe axele x, y, z
        glRotatef(globRotR, 0,0,1);
        glTranslatef(5.0f,0.0f,0.0f);
        glRotatef(axisRot,0,1,0); //rotit pe axa Y
        glutSolidCube(2); //cub cu latura 2
    glPopMatrix();

    glColor3f(0.0f, 1.0f, 0.0f);
    glPushMatrix();
        glTranslatef(0.0f,0.0f,-20); //deplasat pe axele x, y, z
        glRotatef(globRotG, 0,0,1);
        glTranslatef(5.0f,0.0f,0.0f);
        glRotatef(axisRot,0,1,0); //rotit pe axa Y
        glutSolidCube(2); //cub cu latura 2
    glPopMatrix();

    glColor3f(0.0f, 0.0f, 1.0f);
```

```
    glPushMatrix();
        glTranslatef(0.0f,0.0f,-20); //deplasat pe axele x, y, z
        glRotatef(globRotB, 0,0,1);
        glTranslatef(5.0f,0.0f,0.0f);
        glRotatef(axisRot,0,1,0); //rotit pe axa Y
        glutSolidCube(2); //cub cu latura 2
    glPopMatrix();

    axisRot += 1.0f; axisRot=fmod(axisRot, 360.0f);
    globRotR += 0.5f; globRotR=fmod(globRotR, 360.0f);
    globRotG += 0.5f; globRotG=fmod(globRotG, 360.0f);
    globRotB += 0.5f; globRotB=fmod(globRotB, 360.0f);

    glutSwapBuffers();
}

static void
idle(void)
{
    glutPostRedisplay();
}

/* Punct de intrare in program */

int
main(int argc, char *argv[])
{
    int width = 640;
    int height = 480;
    glutInit(&argc, argv);
    glutInitWindowSize(width,height);
    glutInitWindowPosition(10,10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    glutCreateWindow(""); //titlu vid, il setez in display()
    glutDisplayFunc(display);
    glutIdleFunc(idle);
    initGL(width, height);
    glutMainLoop();
    return EXIT_SUCCESS;
}
```

**Exercitiul 1. Compilati si executati codul prezentat mai sus. Incercati sa schimbati parametrii functiilor, experimentati.**

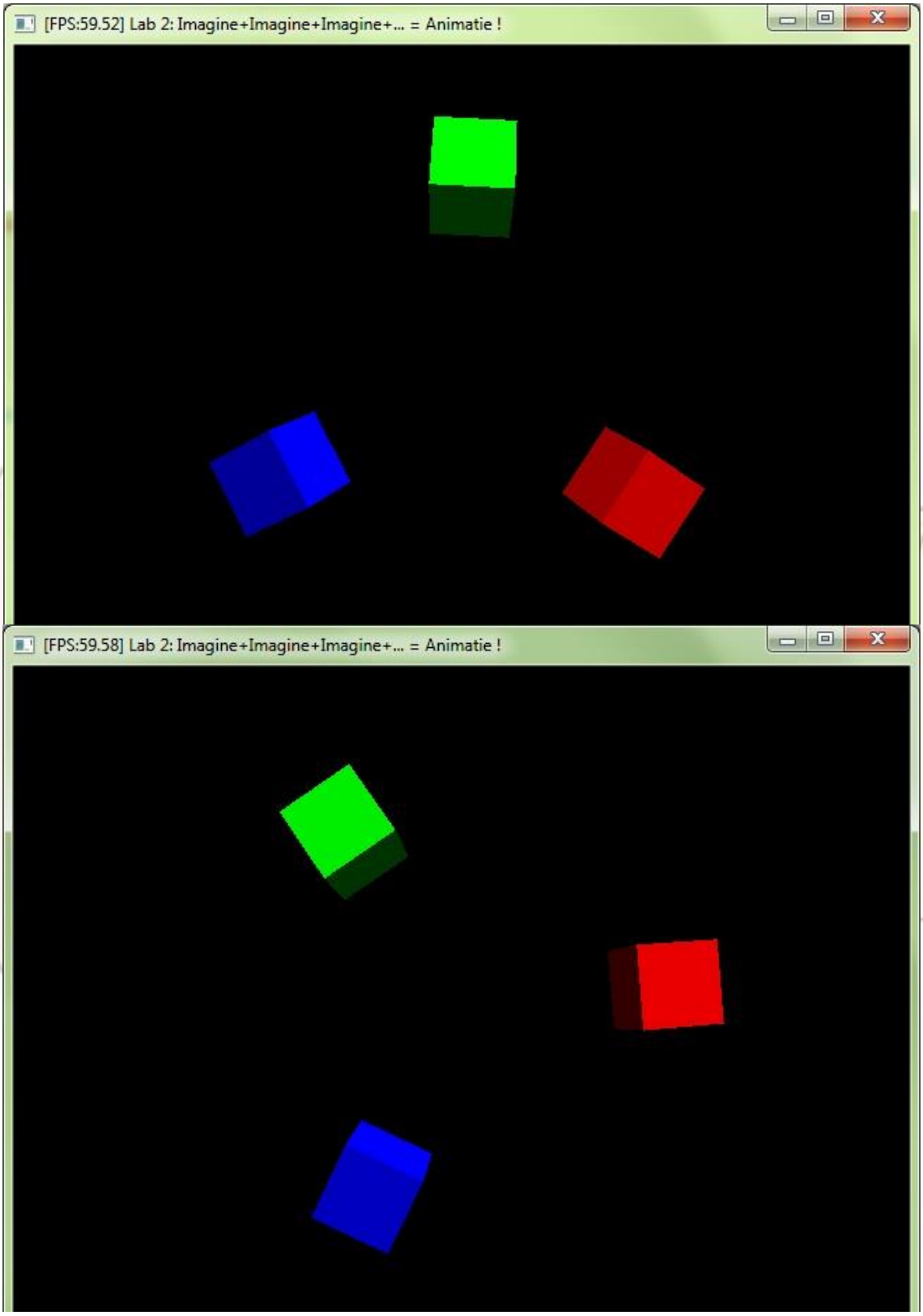


Fig. 1. Prima noastra animatie !

## Elementele din program ce determina animatia

In primul rand ne vor trebui variabile care sa contina deplasarea pana in momentul current.

```
    static float axisRot = 0.0f;
    static float globRotR = 0.0f;
    static float globRotG = 120.0f;
    static float globRotB = 240.0f;
```

Astfel avem o variabila ce retine rotatia in jurul axei proprii (axisRot), si 3 variabile ce retin rotatiile fiecarui cub in jurul originii (globRotR, G, B). Arbitrar, am atribuit un caracter static acestor variabile; acest lucru face ca variabilele marcate cu static sa isi pastreze valorile de la o iteratie la alta. Acelasi efect se putea obtine si cu variabile globale.

Urmeaza afisarea celor 3 cuburi:

```
    glColor3f(1.0f, 0.0f, 0.0f);
    glPushMatrix();
        glTranslatef(0.0f,0.0f,-20); //deplasat pe axele x, y, z
        glRotatef(globRotR, 0,0,1);
        glTranslatef(5.0f,0.0f,0.0f);
        glRotatef(axisRot,0,1,0); //rotit pe axa Y
        glutSolidCube(2); //cub cu latura 2
    glPopMatrix();

    glColor3f(0.0f, 1.0f, 0.0f);
    glPushMatrix();
        glTranslatef(0.0f,0.0f,-20); //deplasat pe axele x, y, z
        glRotatef(globRotG, 0,0,1);
        glTranslatef(5.0f,0.0f,0.0f);
        glRotatef(axisRot,0,1,0); //rotit pe axa Y
        glutSolidCube(2); //cub cu latura 2
    glPopMatrix();

    glColor3f(0.0f, 0.0f, 1.0f);
    glPushMatrix();
        glTranslatef(0.0f,0.0f,-20); //deplasat pe axele x, y, z
        glRotatef(globRotB, 0,0,1);
        glTranslatef(5.0f,0.0f,0.0f);
        glRotatef(axisRot,0,1,0); //rotit pe axa Y
        glutSolidCube(2); //cub cu latura 2
    glPopMatrix();
```

Este important de observat modul in care sunt aplicate transformarile pe fiecare cub in parte:

- prima data cubul este deplasat cu -20 pe axa Z astfel incat sa fie vizibil
- apoi se roteste cubul la un unghi
- datorita rotatiei, translatia aplicata (+5 pe axa X) va fi conform orientarii obiectului
- inainte de a fi afisat cubul se aplica si o rotatie in jurul axei sale

Dupa afisarea celor 3 cuburi, urmeaza pasul de modificare a gradelor de rotatie folosite.

```
    axisRot += 1.0f; axisRot=fmod(axisRot, 360.0f);
    globRotR += 0.5f; globRotR=fmod(globRotR, 360.0f);
    globRotG += 0.5f; globRotG=fmod(globRotG, 360.0f);
    globRotB += 0.5f; globRotB=fmod(globRotB, 360.0f);
```

Fiind variabile statice, acestea isi pastreaza valorile de la o iteratie la alta. Mai exact, aplicand o incrementare (axisRot += 1.0f) obtinem o noua rotatie care difera de vechea rotatie cu un grad. Astfel obtinem iluzia de o miscare continua.

Functia fmod este echivalentul operatorului **%**, dar actioneaza asupra variabile cu virgula flotanta. Mai exact, fmod(a, b) returneaza restul (in virgula flotanta) a impartirii lui a la b. In acest exemplu ne ajuta sa pastram variabilele in intervalul [0, 360) grade.

## Pozitie, Viteza, Acceleratie

In timp ce **pozitia** semnifica unde se afla un obiect intr-un spatiu dat, relativ la un punct de referinta, **viteza** ne spune modul in care se modifica aceasta pozitie in timp.

Pentru a simula o miscare uniforma a unui obiect, este necesara adaugarea in mod constant a unei valori la pozitia curenta, mai exact aplicarea vitezei.

```
Exemplu:

x0 = 0 m //pozitia in momentul t=0, este de 0 metrii
vx = 1 m/s //viteza este de 1 metru pe secunda

-----------------------------------------------------

t=1, (dt = 1-0 = 1):
     x1 = x0 + vx*dt = 0 + 1 = 1m

-----------------------------------------------------

t=2, (dt = 2-1 = 1):
     x2 = x1 + vx*dt = 1 + 1 = 2m

-----------------------------------------------------

...
```

Asa cum pozitia este descrisa separat pe axele X, Y si respectiv Z, si viteza poate descrisa pe fiecare axa in parte.

Conceptele de pozitie si viteza creaza scenariul de miscare rectilinie uniforma (MRU), in care orice obiect are ca traiectorie o dreapta in spatiu, si se deplaseaza cu viteza constanta.

Pentru a fi mai aproape de un scenariu natural, mai trebuie sa luam in calcul conceptul de **acceleratie**, un modificator al vitezei.

Exemplu:

$x_0$ = 0 m //pozitia in momentul t=0, este de 0 metrii
$vx_0$ = 1 m/s //viteza este de 1 metru pe secunda
ax = 1 $m/s^2$ //acceleratie de (1 metru pe secunda) pe secunda

---

t=1, (dt = 1-0 = 1):
$x_1 = x_0 + vx_0*dt = 0 + 1 = 1m$
$vx_1 = vx_0 + ax*dt = 1 + 1 = 2m/s$

---

t=2, (dt = 2-1 = 1):
$x_2 = x_1 + vx_1*dt = 1 + 2 = 3m$
$vx_2 = vx_1 + ax*dt = 2 + 1 = 3m/s$

---

t=3, (dt = 3-2 = 1):
$x_3 = x_2 + vx_2*dt = 3 + 3 = 6m$
$vx_3 = vx_2 + ax*dt = 3 + 1 = 4m/s$

---

...

Atentie ! formulele nu sunt intocmai corecte ! observati ca daca valoarea lui „dt” creste, atunci miscarea va suferi salturi. Dar din fericire, in aplicatiile noastre „dt” va fi suficient de mic astfel incat aceste salturi nu se vor face simtite, deoarece in acest caz, dt va reprezenta diferenta de timp intre doua apeluri ale functiei display.

Toate aceste variabile pot fi incluse intr-o structura, pentru simplitate:

```
struct MaterialPoint
{
	float x, y, z; //pozitie
	float vx, vy, vz; //viteza
	float ax, ay, az; //acceleratie
};
```

Pentru a simplifica si mai mult problema, adaugam si o functie care se ocupa cu calculul variabilelor, si miscarea obiectului:

```
void moveMe(MaterialPoint *m)
{
    glTranslatef(m->x, m->y, m->z);

    m->vx += m->ax;
```

```
    m->vy += m->ay;
    m->vz += m->az;

    m->x += m->vx;
    m->y += m->vy;
    m->z += m->vz;
}
```

In functia display() vom adauga inca un cub, ce simuleaza miscarea unui pendul:

```
    static MaterialPoint *miscare = new MaterialPoint;
    static bool initMiscare = true;
    if (initMiscare)
    {
        miscare->x = 10.0f; //initializare
        initMiscare = false;
    }

    glColor3f(1.0f, 1.0f, 0.0f);
    glPushMatrix();
        //glRotatef(-globRotR, 0,0,1);
        glTranslatef(0.0f,0.0f,-50); //deplasat pe axele x, y, z
        moveMe(miscare);
        glutSolidCube(2); //cub cu latura 2
    glPopMatrix();

    if (miscare->x > 0.0f)
    {
        miscare->ax = -0.01f;
    }
    else
    {
        miscare->ax = 0.01f;
    }
```

Observati ca nu modificam decat acceleratia obiectului, de restul se ocupa functia de mai devreme.

**Exercitiul 2. Actualizati codul vostru cu modificarea de mai sus si experimentati.**

**Exercitiul 3. Folosind cunostintele din acest laborator, programati un cub (sau orice primitiva) sa se deplaseze pe o traiectorie care descrie un patrat.**

**Exercitiul 4. Programati o primitiva sa se deplaseze pe o traiectorie care descrie o spirala.**

## Interactiunea cu tastatura

Adaugati urmatoarea functie in program:

```
static void
keyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
        case 'a':
            //s-a apasat tasta A
            break;
        case 'b':
            //s-a apasat tasta B
            break;
        case 'c':
            //s-a apasat tasta C
            break;
    }
}
```

In functia main, adaugati urmatorul apel de functie:

```
    glutKeyboardFunc(keyboard);
```

**Exercitiul 5. Folosind interactiunea cu tastatura, adaptati programul sa modifice valoarea acceleratiei prin apasarea tastelor + si - .**

**Exercitiul 6. Programati miscarea cubului de la tastatura (pe axele X si Z, sau la alegere).**