

Lucrarea nr. 4

Elemente de grafică în OpenGL

1. Scopul lucrării:

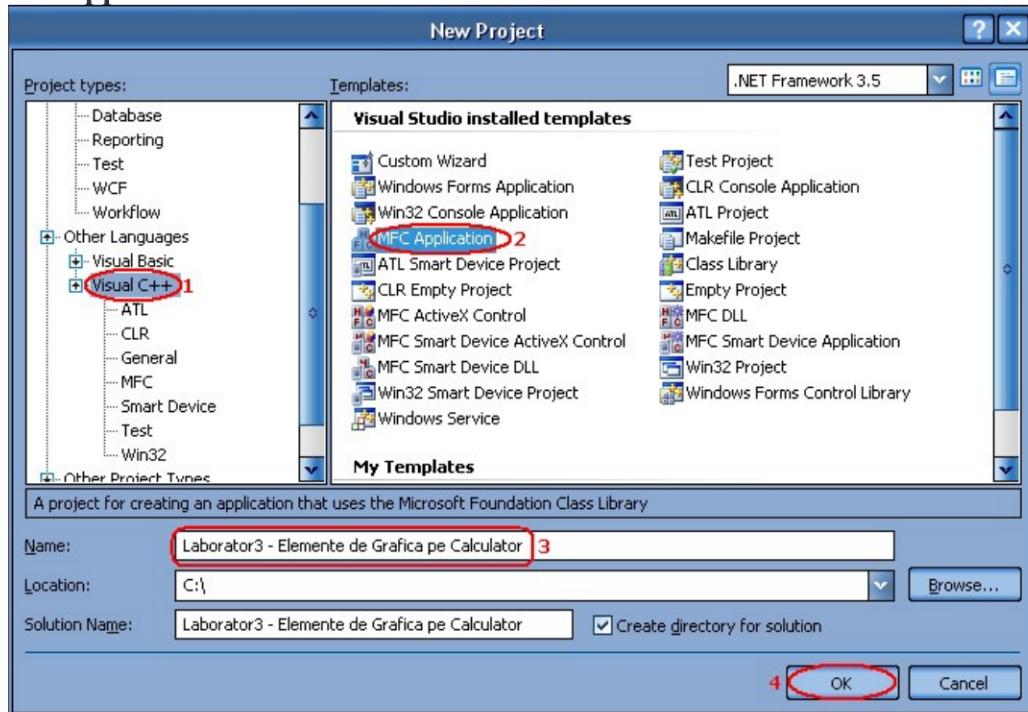
Lucrarea de față își propune prezentarea conceptelor și tehniciilor de bază OpenGL. În această lucrare continuăm prezentarea primitivelor din OpenGL (linii, triunghiuri, patrulatere, poligoane), funcțiile puse la dispoziția programatorilor pentru definirea transformărilor de coordonate și modul de construire al aplicațiilor grafice simple utilizând aceste funcții și primitive.

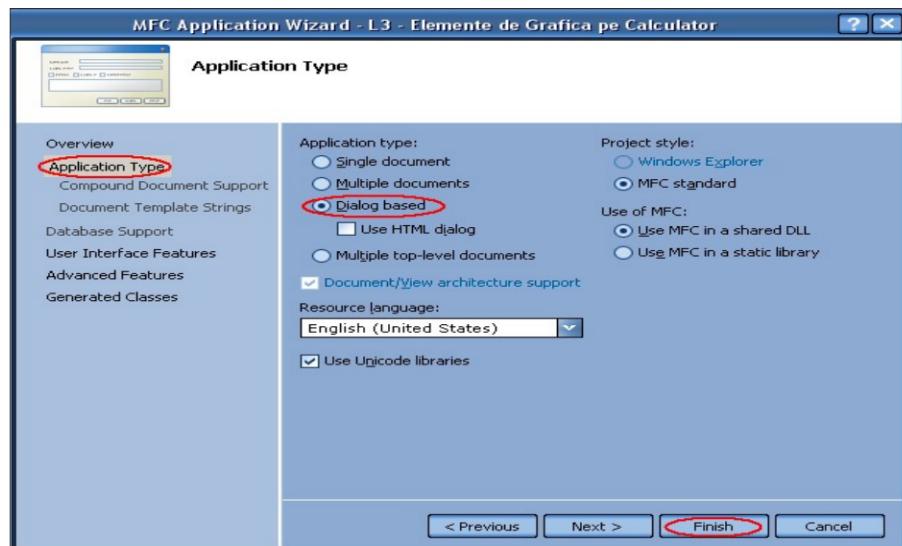
2. Noțiuni teoretice:

OpenGL și MFC

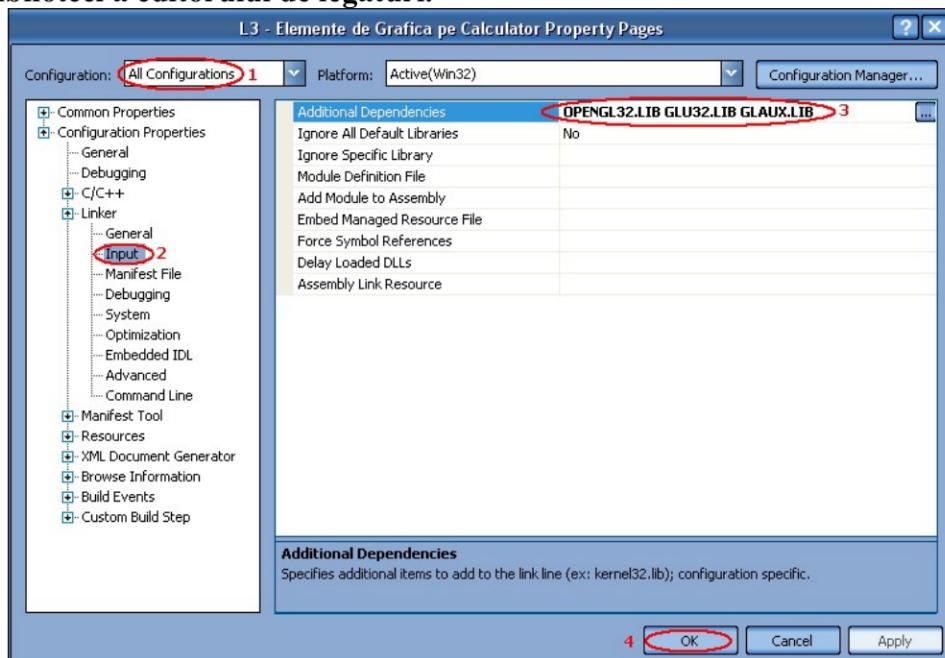
Toate exemplele din acest laborator utilizează biblioteca AUX pentru crearea ferestrei de lucru OpenGL și pentru tratarea evenimentelor de intrare-iesire. Etapele ce trebuie urmate pentru a transforma un context de afișare Windows într-un dispozitiv de randare OpenGL sunt următoarele:

1. Folosind Microsoft Visual Studio, se creează un proiect de tip aplicație MFC AppWizard.





2. Se adaugă bibliotecile: **OPENGL32.lib** **GLU32.lib** **GLAUX.LIB** la lista de biblioteci a editorului de legături.



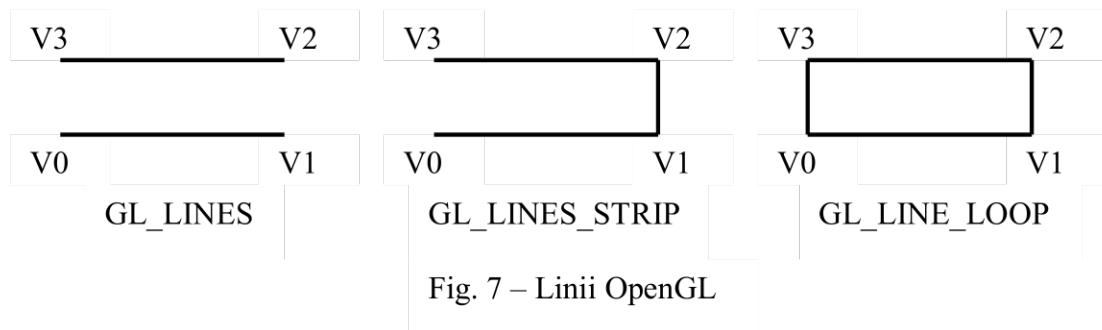
3. Se adaugă în fișierul stdafx.h header-ele:

```
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glaux.h>
```

Desenare linii în OpenGL

Pentru desenarea liniilor și a liniilor poligonale (închise sau deschise) OpenGL oferă trei primitive (*Figura 7*):

- **GL_LINES** pentru desenarea unei colecții de linii.
- **GL_LINE_STRIP** pentru desenarea unei linii poligonale deschise.
- **GL_LINES_LOOP** pentru desenarea unei linii poligonale închise.



Stabilirea grosimii liniei se face utilizând funcția

```
void glLineWidth(GLfloat width);
```

unde width reprezintă grosimea liniei în pixeli. Similar cu dimensiunea punctului, variabilele de stare `GL_LINE_WIDTH_RANGE` și `GL_LINE_WIDTH_GRANULARITY` controlează valorile valide pentru grosimea liniei.

Stabilirea stilului liniei (continuă, punctată, linie-punct, etc.) se face utilizând funcția

```
void glLineStipple(GLint factor, GLushort pattern);
```

unde factor reprezintă numărul de pixeli corespunzători unui bit al şablonului, iar pattern reprezintă şablonul de desenare al liniei (*Figura 8*). Pentru un bit al patternului vor fi “aprinși” factor pixeli, respectiv pentru un bit 0 vor fi “stinsă” factor pixeli. Bitul 0 (cel mai puțin semnificativ) al şablonului este folosit pentru primul pixel al liniei. Şablonul este repetat pe întreaga lungime a liniei. La folosirea stilurilor de linie trebuie activată și variabila de stare `GL_LINE_STIPPLE`.

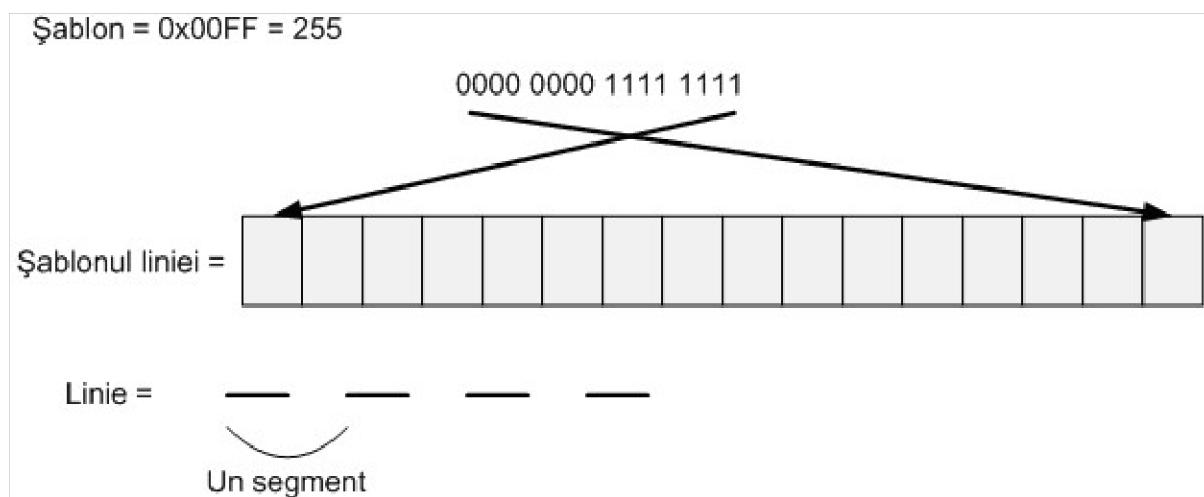


Figura 8 – Şablonul liniei

În exemplul următor, în funcție de tipul primitivei specificate funcției `glBegin`, va fi desenată una din cele trei tipuri de linii (segment de linie, linie poligonală deschisă sau închisă) cu stilul 0x5555.

Exemplu 5: Desenarea liniilor în OpenGL (*Figura 9*)

```
void CALLBACK ModificaDimensiune(GLsizei w, GLsizei h)
{
    //Construirea volumului de vedere
    GLfloat nRange = 100.0f;

    if (h==0) h = 1;
```

```
// Stabilirea viewportului la dimensiunea ferestrei
glViewport(0, 0, w, h);

// Initializeaza matricea de proiectie cu matricea identitate
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

// Stabileste volumul de vedere folosind o proiectie ortografica
if (w<=h)
    glOrtho(-nRange, nRange, -nRange*h/w, nRange*h/w, -nRange, nRange);
else
    glOrtho(-nRange*w/h, nRange*w/h, -nRange, nRange, -nRange, nRange);

// Initializarea matricii modelului
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}

void CALLBACK DeseneazaScena(void)
{
    // Factorul si sablonul pentru stilul liniei
    GLint factor = 10;
    GLushort pattern = 0xffff;
    GLfloat widths[2], gr;

    // Stabileste culoarea liniei
    glColor3f(1.0f, 1.0f, 0.0f);

    // Obtinerea domeniului valid pentru grosime
    glGetFloatv(GL_LINE_WIDTH_RANGE, widths);
    glGetFloatv(GL_LINE_WIDTH_GRANULARITY, &gr);

    // Activeaza stilul la desenarea liniei
    glEnable(GL_LINE_STIPPLE);

    // Stabileste stilul si grosimea liniei
    glLineStipple(factor, pattern);
    glLineWidth(widths[0]+10*gr);

    // Deseneaza primitiva
/*glBegin(GL_LINES);
    glVertex2f(0.0f, 50.0f);
    glVertex2f(50.0f, 50.0f);
    glVertex2f(50.0f, 0.0f);
    glVertex2f(0.0f, 0.0f);
glEnd();*/

    // Deseneaza primitiva
    glBegin(GL_LINES);
    // toate liniile vor fi trasate in planul xy
    z = 0.0f;
    for(angle = 0.0f; angle <= GL_PI; angle += (GL_PI/20.0f))
    {
        // jumatatea de sus a cercului (partea superioara)
        x = 50.0f*sin(angle);
        y = 50.0f*cos(angle);
        glVertex3f(x, y, z);      // primul punct al liniei

        // a doua jumatate a cercului (partea de jos)
        x = 50.0f*sin(angle + GL_PI);
```

```

        y = 50.0f*cos(angle + GL_PI);
        glVertex3f(x, y, z);      // al doilea punct al liniei
    }
    glEnd();

    glFlush();
}

void CTeste_OpenGLDlg::OnBnClickedButton1()
{
    // Stabilirea ferestrei de lucru OpenGL
    auxInitDisplayMode(AUX_SINGLE|AUX_RGB);
    auxInitPosition(10, 10, 500, 500);
    auxInitWindow("Linii");

    // Înregistrarea functiilor CALLBACK
    auxReshapeFunc(ModificaDimensiune);
    auxMainLoop(DeseneazaScena);
}

```

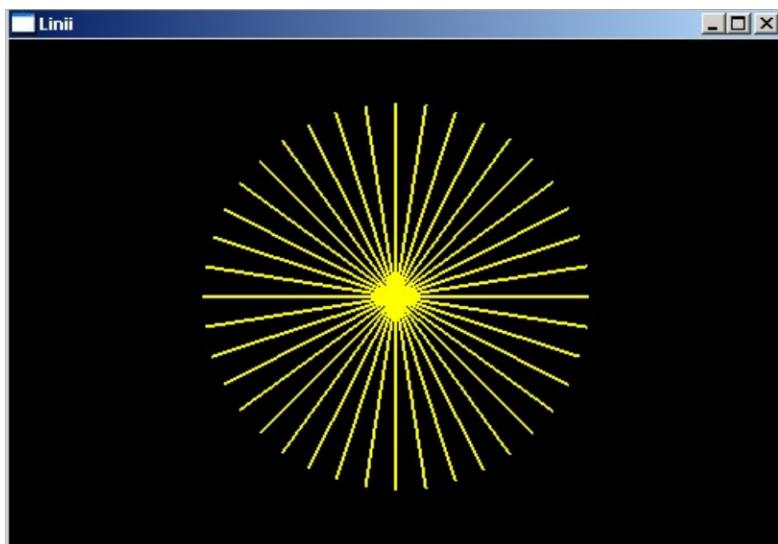


Figura 9 – Linii în OpenGL

Exemplul 6: Desenarea liniilor în OpenGL

```

//Față de exemplul anterior, în funcția void CALLBACK
//DeseneazaScena(void) se înlocuiește primitiva glBegin(GL_LINES) cu
//primitivile glBegin(GL_LINE_STRIP); (Fig. 10a) și glBegin
//(GL_LINE_STRIP); (Fig. 10b).

glBegin(GL_LINE_STRIP);
    glVertex3f(0.0f, 0.0f, 0.0f); // v0
    glVertex3f(50.0f, 50.0f, 0.0f); // v1
    glVertex3f(50.0f, 100.0f, 0.0f); // v2
glEnd();

glBegin(GL_LINE_LOOP);
    glVertex3f(0.0f, 0.0f, 0.0f); // v0
    glVertex3f(50.0f, 50.0f, 0.0f); // v1
    glVertex3f(50.0f, 100.0f, 0.0f); // v2
glEnd();

```

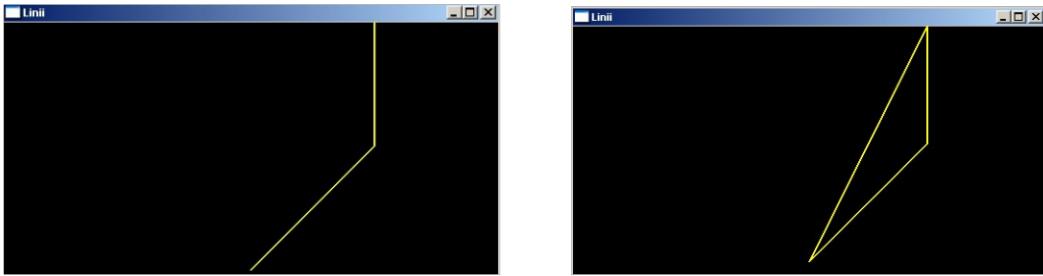


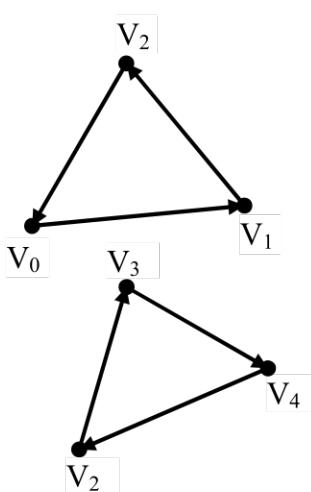
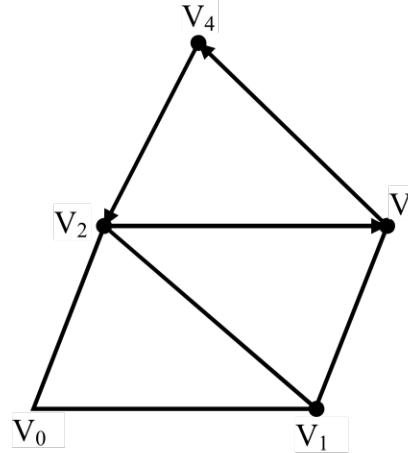
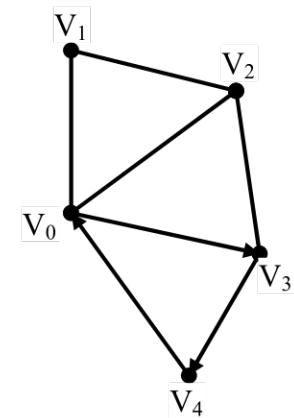
Figura 10

- a) Linii poligonale deschise
- b) Linii poligonale închise

Desenare triunghiuri în OpenGL

Pentru desenarea triunghiurilor, OpenGL oferă trei primitive (*Figura 1*):

- **GL_TRIANGLES** pentru desenarea unei colecții de triunghiuri.
- **GL_TRIANGLE_STRIP** pentru desenarea unui “lanț” de triunghiuri.
- **GL_TRIANGLE_FAN** pentru desenarea unui evantai compus din triunghiuri.

**GL_TRIANGLES****GL_TRIANGLE_STRIP****GL_TRIANGLE_FAN****Figura 1 – Triunghiuri OpenGL**

Un aspect deosebit de important la definirea triunghiurilor folosind primitiva **GL_TRIANGLES** este ordinea în care sunt specificate vârfurile triunghiului. Această ordine definește **orientarea triunghiului (winding)**, iar în general orientarea unui poligon cu n vârfuri, și poate fi de două tipuri:

- în sensul acelor de ceasornic.
- în sens trigonometric (contrar acelor de ceasornic).

În mod implicit, OpenGL consideră poligoanele cu orientare trigonometrică ca fiind cu față la privitor. Acest lucru este important deoarece celor două fețe ale unui poligon (identificate prin constantele **GL_FRONT** și **GL_BACK**) li se pot stabili proprietăți fizice diferite (mod de desenare, culoare, textură, etc.). Orientarea triunghiurilor poate fi modificată cu ajutorul următorului apel:

```
glFrontFace(GL_CW);
```

În cazul primitivelor GL_TRIANGLE_STRIP și GL_TRIANGLE_FAN vârfurile sunt rearanjate de OpenGL astfel încât toate triunghiurile sunt parcuse în sens trigonometric.

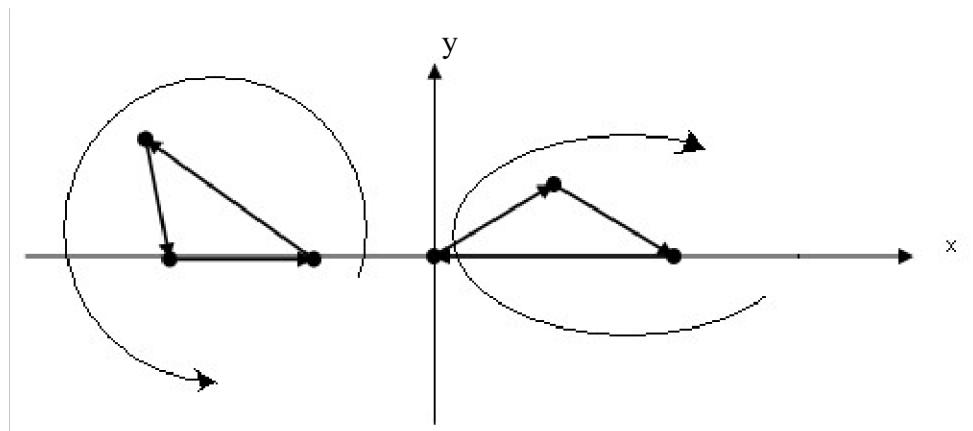


Figura 2 – Orientarea triunghiurilor

Desenare patrulatere și poligoane în OpenGL

Pentru desenarea patrulaterelor OpenGL pune la dispoziție două primitive (Figura 3):

- **GL_QUADS** pentru desenarea unui patrulater.
- **GL_QUAD_STRIP** pentru desenarea unui „lanț” de patrulare.

Pentru desenarea poligoanelor cu un număr oarecare de vârfuri se folosește primitiva **GL_POLYGON** (Figura 3). În OpenGL se pot construi doar poligoane convexe, adică poligoane ale căror vârfuri sunt coplanare, laturile nu se intersecțează și toate vârfurile se află de aceeași parte a unei laturi (testul de convexitate). Toate cele trei primitive au o orientare în sensul acelor de ceasornic. Poligoanele neconvexe pot fi desenate prin descompunerea lor în două sau mai multe poligoane convexe. Pentru a ascunde muchiile de îmbinare între poligoane interne poligonului final, se folosește funcția:

```
glEdgeFlag (Glboolean flag);
```

care indică dacă muchiile definite de următoarele vârfuri sunt muchii de contur (true) sau nu (false).

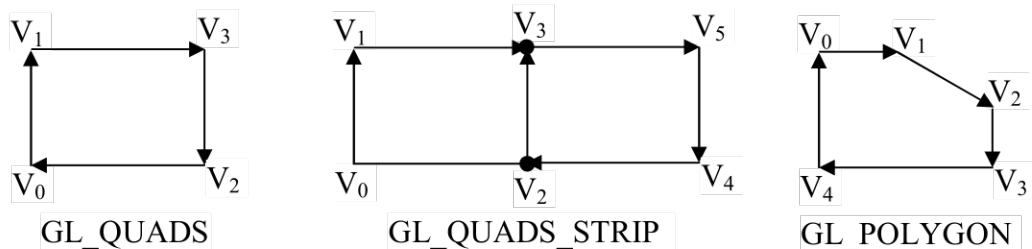


Figura 3 – Patrulatere și poligoane OpenGL

Pentru umplerea poligoanelor cu un şablon vom folosi funcția:

```
 glEnable (GL_POLYGON_STIPPLE);
 glPolygonStipple (Glubyte* pattern);
```

unde pattern este un tablou bidimensional de 32×32 biți în care fiecare bit 1 corespunde unui pixel aprins, în timp ce biții 0 corespund pixelilor stinși. Şablonul este repetat pe întreaga suprafață a poligonului și este parcurs de jos în sus și de la stânga la dreapta.

Exemplul 1: Desenarea unui hexagon umplut cu un şablon (Figura 4)

```

void CL2_1Dlg::OnBnClickedButton1()
{
    // Stabilirea ferestrei de lucru OpenGL
    auxInitDisplayMode(AUX_SINGLE|AUX_RGB);
    auxInitPosition(10, 10, 500, 500);
    auxInitWindow("Hexagon umplut cu un sablon");

    // Înregistrarea functiilor CALLBACK
    auxReshapeFunc(ModificaDimensiune);
    auxMainLoop(DeseneazaScena);
}

void CALLBACK ModificaDimensiune(GLsizei w, GLsizei h) //Construirea
volumului de vedere
{
    GLfloat nRange = 100.0f;

    if (h==0) h = 1;

    // Stabilirea viewportului la dimensiunea ferestrei
    glViewport(0, 0, w, h);

    // Initializeaza matricea de proiectie cu matricea identitate
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // Stabileste volumul de vedere folosind o proiectie ortografica
    if (w<=h)
        glOrtho(-nRange, nRange, -nRange*h/w, nRange*h/w, -nRange, nRange);
    else
        glOrtho(-nRange*w/h, nRange*w/h, -nRange, nRange, -nRange, nRange);

    // Initializarea matricii modelului
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void CALLBACK DeseneazaScena(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    // Specificare sablonului
    glEnable(GL_POLYGON_STIPPLE);
    glPolygonStipple(fly);

    // Stabilirea culorii de desenare
    glColor3f(1.0f, 0.0f, 0.0f);

    // Definirea poligonului
    glBegin(GL_POLYGON);
        glVertex2f(-20.0f, 50.0f);
        glVertex2f(20.0f, 50.0f);
        glVertex2f(50.0f, 20.0f);
        glVertex2f(50.0f, -20.0f);
        glVertex2f(20.0f, -50.0f);
}

```

```

        glVertex2f(-20.0f, -50.0f);
        glVertex2f(-50.0f, -20.0f);
        glVertex2f(-50.0f, 20.0f);
    glEnd();
    glFlush();
}

// Definirea şablonului
GLubyte fly[] = {
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x03, 0x80, 0x01, 0xC0,
    0x06, 0xC0, 0x03, 0x60,
    0x04, 0x60, 0x06, 0x20,
    0x04, 0x30, 0x0C, 0x20,
    0x04, 0x18, 0x18, 0x20,
    0x04, 0x0C, 0x30, 0x20,
    0x04, 0x06, 0x60, 0x20,
    0x44, 0x03, 0xC0, 0x22,
    0x44, 0x01, 0x80, 0x22,
    0x66, 0x01, 0x80, 0x66,
    0x33, 0x01, 0x80, 0xCC,
    0x19, 0x81, 0x81, 0x98,
    0x0C, 0xC1, 0x83, 0x30,
    0x07, 0xe1, 0x87, 0xe0,
    0x03, 0x3f, 0xfc, 0xc0,
    0x03, 0x31, 0x8c, 0xc0,
    0x03, 0x33, 0xcc, 0xc0,
    0x06, 0x64, 0x26, 0x60,
    0x0c, 0xcc, 0x33, 0x30,
    0x18, 0xcc, 0x33, 0x18,
    0x10, 0xc4, 0x23, 0x08,
    0x10, 0x63, 0xC6, 0x08,
    0x10, 0x30, 0x0c, 0x08,
    0x10, 0x18, 0x18, 0x08,
    0x10, 0x00, 0x00, 0x08};

GLubyte fire[] = {
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0xc0,
    0x00, 0x00, 0x01, 0xf0,
    0x00, 0x00, 0x07, 0xf0,
    0x0f, 0x00, 0x1f, 0xe0,
    0x1f, 0x80, 0x1f, 0xc0,
    0x0f, 0xc0, 0x3f, 0x80,
    0x07, 0xe0, 0x7e, 0x00,
    0x03, 0xf0, 0xff, 0x80,
    0x03, 0xf5, 0xff, 0xe0,
    0x07, 0xfd, 0xff, 0xf8,
    0x1f, 0xfc, 0xff, 0xe8,
    0xff, 0xe3, 0xbf, 0x70,
}

```

```

    0xde, 0x80, 0xb7, 0x00,
    0x71, 0x10, 0x4a, 0x80,
    0x03, 0x10, 0x4e, 0x40,
    0x02, 0x88, 0x8c, 0x20,
    0x05, 0x05, 0x04, 0x40,
    0x02, 0x82, 0x14, 0x40,
    0x02, 0x40, 0x10, 0x80,
    0x02, 0x64, 0x1a, 0x80,
    0x00, 0x92, 0x29, 0x00,
    0x00, 0xb0, 0x48, 0x00,
    0x00, 0xc8, 0x90, 0x00,
    0x00, 0x85, 0x10, 0x00,
    0x00, 0x03, 0x00, 0x00,
    0x00, 0x00, 0x10, 0x00
};

};


```



Figura 4 – Hexagon umplut cu un şablon

Primitivele de tip poligon (triunghi, patrulater, poligon) sunt desenate implicit “pline”, dar acestea pot fi desenate și în cadru de sârmă sau prin puncte folosind funcția

```
void glPolygonMode(GLenum face, GLenum mode);
```

Primul argument specifică fața poligonului căreia i se aplică noul mod de desenare (GL_FRONT, GL_BACK sau GL_FRONT_AND_BACK), iar al doilea specifică modul de desenare: GL_FILL (plin), GL_LINE (cadru de sârmă) sau GL_POINT (doar vîrfurile poligonului).

În OpenGL, culoarea se specifică pentru fiecare vîrf al poligonului. Modelul de umbrire folosit (shading model) determină dacă poligonul este colorat uniform cu culoarea specificată pentru ultimul său vîrf (GL_FLAT) sau dacă culorile sunt interpolate între vîrfurile poligonului (GL_SMOOTH). Stabilirea modelului de umbrire se face cu funcția:

```
void glShadeModel (GLenum mode);
```

Exemplul 2: Desenare triunghi în OpenGL cu vîrfurile de culori diferite folosind interpolarea (Figura 5)

```

void CL2_1Dlg::OnBnClickedButton1()
{
    // Stabilirea ferestrei de lucru OpenGL
    auxInitDisplayMode(AUX_SINGLE|AUX_RGB);
    auxInitPosition(10, 10, 500, 500);
    auxInitWindow("Poligon cu sablon");

    // Înregistrarea functiilor CALLBACK
}


```

```

    auxReshapeFunc (ModificaDimensiune) ;
    auxMainLoop (DeseneazaScena) ;
}

void CALLBACK DeseneazaScena (void)
{
    //glShadeModel (GL_SMOOTH) ;
    glShadeModel (GL_SMOOTH) ;
    glBegin (GL_TRIANGLES) ;
        // varful rosu
        glColor3f (1.0f, 0.0f, 0.0f) ;
        glVertex2f (-50.0f, 0.0f) ;

        // varful verde
        glColor3f (0.0f, 1.0f, 0.0f) ;
        glVertex2f (0.0f, 70.0f) ;

        // varful albastru
        glColor3f (0.0f, 0.0f, 1.0f) ;
        glVertex2f (50.0f, 0.0f) ;
    glEnd () ;
    glFlush () ;
}

void CALLBACK ModificaDimensiune (GLsizei w, GLsizei h) //Construirea
volumului de vedere
{
    GLfloat nRange = 100.0f;

    if (h==0) h = 1;

    // Stabilirea viewportului la dimensiunea ferestrei
    glViewport (0, 0, w, h);

    // Initializeaza matricea de proiectie cu matricea identitate
    glMatrixMode (GL_PROJECTION) ;
    glLoadIdentity () ;

    // Stabileste volumul de vedere folosind o proiectie ortografica
    if (w<=h)
        glOrtho (-nRange, nRange, -nRange*h/w, nRange*h/w, -nRange, nRange) ;
    else
        glOrtho (-nRange*w/h, nRange*w/h, -nRange, nRange, -nRange, nRange) ;

    // Initializarea matricii modelului
    glMatrixMode (GL_MODELVIEW) ;
    glLoadIdentity () ;
}

```

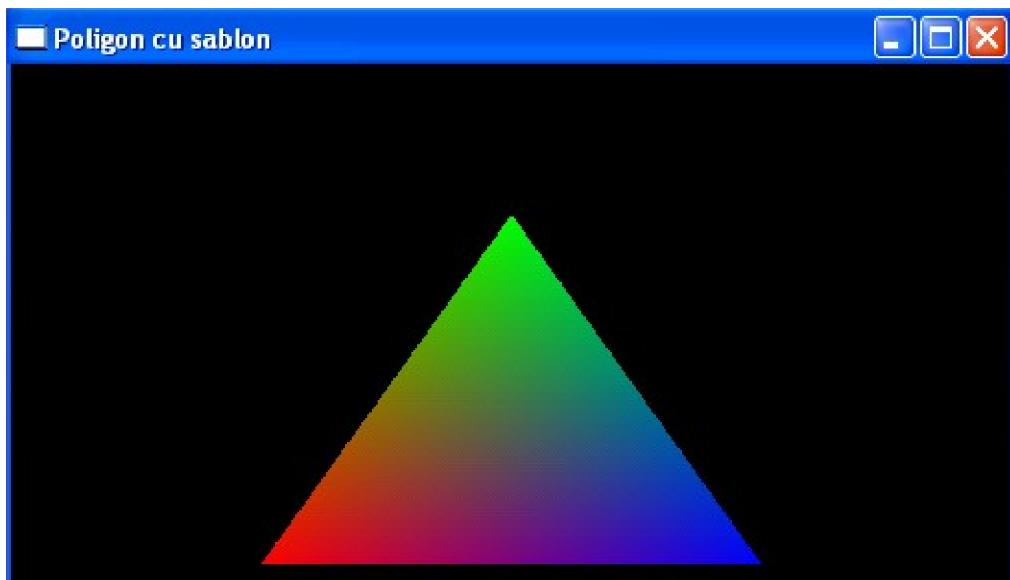


Figura 5 – Triunghi în OpenGL

Pentru ascunderea suprafețelor invizibile trebuie activat testul de adâncime și inițializat bufferul de adâncime înaintea randării scenei. Dacă nu este activat testul de adâncime, întotdeauna ultima primitivă desenată va acoperi primitivele desenate anterior, chiar dacă acestea se află mai aproape de privitor. Lucrul în mod test de adâncime se face astfel:

```
glEnable(GL_DEPTH_TEST);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
```

Operarea în mod test de adâncime este mai lentă decât cea implicită. Pentru a mări viteza de randare în acest caz se pot elimina calculele efectuate de OpenGL pentru partea din față/spate a fiecărui poligon atunci când știm *a priori* că această parte nu este vizibilă în obiectul randat. Acest mod de lucru se numește *culling* și este controlat de variabila de stare GL_CULL_FACE.

Anti-aliasing

Desenarea primitivelor în OpenGL poate introduce defecte nedorite (numite *aliasing*) datorate discretizării imaginilor (ex.: linii înclinate în formă de scăriță). OpenGL pune la dispoziție două tehnici de combatere a acestor defecte (numite tehnici pentru anti-aliasing):

- prin combinarea culorilor (blending)
- prin utilizarea bufferului de acumulare.

În continuare vom prezenta prima tehnică – combinarea culorilor. **Metoda combinării culorilor** dă rezultate numai dacă se dezactivează testul de adâncime, prin apelul glDisable(GL_DEPTH_BUFFER_TEST). În această situație, culoarea unui pixel acoperit parțial de un poligon se obține prin combinarea culorii poligonului cu culoarea pixelului aflat în bufferul de culoare. În modul RGBA, se multiplică componenta alpha (A) a culorii poligonului cu ponderea de acoperire (raportul dintre aria acoperită de poligon și aria pixelului). Această valoare poate fi folosită pentru ponderarea culorii pixelului prin combinare cu factorul GL_SRC_ALPHA pentru sursă, respectiv GL_ONE_MINUS_SRC_ALPHA pentru destinație. Pentru efectuarea acestor calcule trebuie validat modul de lucru anti-aliasing

pentru primitiva care este desenată, înainte de desenarea primitivei. Acesta se face prin activarea variabilelor de stare GL_POINT_SMOOTH (pentru puncte), GL_LINE_SMOOTH (pentru linii) sau GL_POLYGON_SMOOTH (pentru poligoane). În exemplul 3 se desenează un segment de linie folosind tehnica anti-aliasing descrisă mai sus.

Exemplul 3: Anti-aliasing prin combinarea culorilor (Figura 6)

```

void CL2_1Dlg::OnBnClickedButton1()
{
    // Stabilirea ferestrei de lucru OpenGL
    auxInitDisplayMode(AUX_SINGLE|AUX_RGB);
    auxInitPosition(10, 10, 500, 500);
    auxInitWindow("Antialiasing - metoda combinarii culorilor");

    // Înregistrarea functiilor CALLBACK
    auxReshapeFunc(ModificaDimensiune);
    auxMainLoop(DeseneazaLinie);
}

void CALLBACK DeseneazaLinie(void)
{
    // Linie poligonala cu "defecte" (fara anti-aliasing)
    glBegin(GL_LINE_LOOP);
        glVertex2d(-50, 0);
        glVertex2d(0, -70);
        glVertex2d(70, 0);
    glEnd();

    // Activare anti-aliasing
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable(GL_BLEND);
    glEnable(GL_LINE_SMOOTH);

    // Linie poligonala fara "defecte" (folosind anti-aliasing)
    glBegin(GL_LINE_LOOP);
        glVertex2d(-50, 0);
        glVertex2d(0, 70);
        glVertex2d(70, 0);
    glEnd();
    glFlush();
}

void CALLBACK ModificaDimensiune(GLsizei w, GLsizei h) //Construirea
volumului de vedere
{
    GLfloat nRange = 100.0f;

    if (h==0) h = 1;

    // Stabilirea viewportului la dimensiunea ferestrei
    glViewport(0, 0, w, h);

    // Initializeaza matricea de proiectie cu matricea identitate
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // Stabileste volumul de vedere folosind o proiectie ortografica
    if (w<=h)
        glOrtho(-nRange, nRange, -nRange*h/w, nRange*h/w, -nRange, nRange);
    else
}

```

```

glOrtho(-nRange*w/h,nRange*w/h,-nRange,nRange,-nRange,nRange);

// Initializarea matricii modelului
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}

```

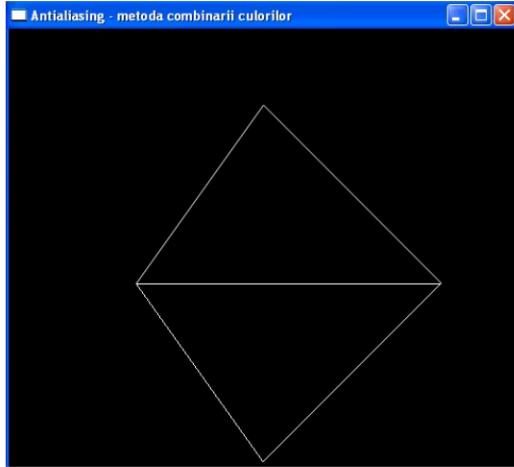


Figura 6 – Linii în OpenGL cu și fără anti-aliasing

Transformări de coordonate

Transformările de coordonate fac posibilă plasarea și orientarea obiectelor în scenă după dorința programatorului. Aceste transformări sunt aplicate sistemului de coordonate și nu unui obiect în parte. Obiectele sunt plasate și rotite ca urmare a desenării lor în sistemul de coordonate astfel transformat. OpenGL implementează o transformare de coordonate printr-o matrice de dimensiune 4×4 , numită **matricea transformării**.

Pentru cumularea transformărilor de coordonate și pentru aplicarea acestora unui vârf al unei primitive în vederea proiecțării lui pe viewport, motorul de randare OpenGL utilizează două matrici de dimensiune 4×4 :

- matricea modelului (modelview matrix)
- matricea proiecție (projection matrix).

După cum se observă în Figura 7, unui vârf i se aplică o serie de transformări până când acesta „ajunge” pe ecran. Aplicarea unei transformări se traduce printr-o operație de înmulțire de matrici sau înmulțirea unui vector cu o matrice. **Matricea modelului** cumulează toate transformările geometrice (translații, rotații, scalări), transformările de vedere (folosind funcția gluLookAt) și alte transformări particulare definite de utilizator (folosind funcții de nivel jos pentru lucrul cu matrici în OpenGL) asupra sistemului de coordonate al lumii reale (cel în care sunt precizate obiectele scenei). **Matricea de proiecție** precizează tipul proiecției și este stabilită fie folosind funcții de nivel înalt (glOrtho sau gluPerspective) fie utilizând funcțiile de nivel jos pentru lucrul cu matrici.

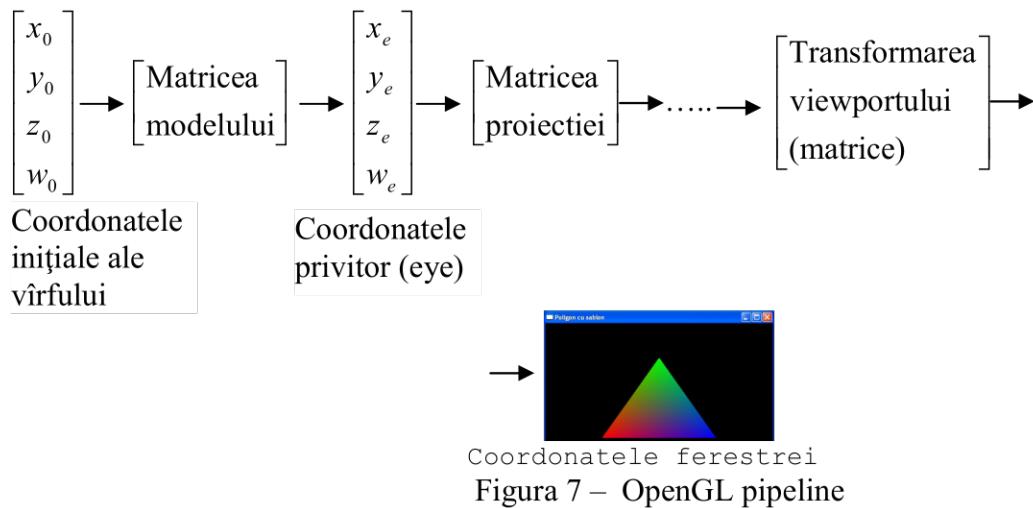


Figura 7 – OpenGL pipeline

Standardul OpenGL pune la dispoziția programatorului trei funcții de nivel înalt (`glTranslate`, `glRotate` și `glScale`) pentru definirea transformărilor geometrice. Aceste funcții operează asupra matricii modelului (MM) după următoarea formulă:

$$MM = MM * MT$$

unde MT este matricea transformării (translației, rotației sau scalării) și este calculată automat de funcție. În continuare sunt prezentate variantele cu argumente flotante pentru aceste funcții:

```
void glTranslatef (GLfloat x, GLfloat y, GLfloat z);
void glRotatef (GLfloat angle, GLfloat x, GLfloat y, GLfloat z);
void glScalef(GLfloat x, GLfloat y, GLfloat z);
```

unde x, y și z sunt valorile pe cele 3 axe, iar *angle* este unghiul de rotație precizat în grade.

Deoarece operația de înmulțire a matricilor nu este comutativă, ordinea în care sunt aplicate transformările sistemului de coordonate este foarte importantă. De cele mai multe ori efectul unei translații urmat de o rotație diferă de cel al rotației urmat de translație. De exemplu, execuția codului:

```
glTranslatef (10,10, 0);
glRotatef (90, 1, 0, 0);
```

nu este echivalentă cu cea a codului:

```
glRotatef (90, 1, 0, 0);
glTranslatef (10,10, 0);
```

Funcțiile de nivel jos pentru lucrul cu matrici în OpenGL operează asupra matricii de lucru actuale și sunt următoarele:

- `glMatrixMode(GLenum mode)`; stabilește matricea de lucru (mode poate lua una din valorile `GL_MODELVIEW`, `GL_PROJECTION`, `GL_TEXTURE`).
- `glLoadIdentity()`; încarcă matricea identitate de dimensiune 4*4 în matricea de lucru și are ca efect restaurarea sistemului de coordonate inițial.
- `glLoadMatrixf (const GLfloat* m)`; încarcă matricea m în matricea de lucru.
- `glMultMatrixf (const GLfloat* m)`; înmulțește matricea m cu matricea de lucru.

Deși funcția `glLoadIdentity` ne oferă posibilitatea să reinițializăm sistemul de coordonate înainte de poziționarea fiecărui obiect, acest lucru nu este întotdeauna de ajuns și nici dorit (de exemplu, în cazul rotirii întregii scene cu un anumit unghi sau aplicării unei transformări unui grup de obiecte, o modificare a sistemului de coordonate înainte de poziționarea fiecărui obiect al grupului (scenei) ar anula transformarea generală). Mai util ar fi un mecanism de salvare a

transformării curente înainte de poziționarea unui obiect și de restaurare a ei după desenarea obiectului. Acest lucru este implementat în OpenGL cu ajutorul **stivelor de matrici** care sunt tablouri de matrici ce funcționează pe principiul ultimul – intrat – primul – ieșit și pot fi manipulate cu ajutorul funcțiilor:

- glPushMatrix(); - salvează matricea de lucru actuală pe stivă
- glPopMatrix(); - restaurează matricea de lucru din vârful stivei

Există 3 stive de matrici în OpenGL:

- stiva modelului, în care se salvează matricea modelului.
- stiva proiecției, în care se salvează matricea proiecției.
- stiva texturării, în care se salvează matricea texturii..

Înainte de apelul funcțiilor de mai sus trebuie stabilită matricea de lucru (cu ajutorul funcției glMatrixMode) care va indica asupra cărei stive se va acționa.

Exemplul 4: Manipulare obiecte (triunghiuri) in OpenGL folosind translatii, rotatii si scalari (Figura 8)

```

GLfloat fTranslate;           // variabila folosita la translatie
GLfloat fRotate;             // variabila folosita la rotatie
GLfloat fScale    = 1.0f;     // variabila folosita la scalare - implicit 1

void Draw_Triangle();         // Declaratie functie ce deseneaza triunghi

void CALLBACK ModificaDimensiune(GLsizei width, GLsizei height)
{   //redimensionare scena
    if (height==0)      // Prevenim impartirea la 0
        height=1;
// Stabilirea viewportului la dimensiunea ferestrei
    glViewport(0,0,width,height);
    glMatrixMode(GL_PROJECTION); // Selectare Matrice Proiectie
    glLoadIdentity();          // Resetare Matrice Proiectie

// Stabileste volumul de vedere folosind o proiectie perspectiva
    gluPerspective(45.0f, (GLfloat)width/(GLfloat)height, 0.1f,100.0f);

    glMatrixMode(GL_MODELVIEW); // Selectare Matrice ModelView
    glLoadIdentity();          // Resetare Matrice Proiectie
}

//Initializare OpenGL - toate setarile OpenGL se fac in aceasta functie
int InitGL(GLvoid)
{
    glShadeModel(GL_SMOOTH);           // Stabilire model umbrire
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // fundal negru
    glClearDepth(1.0f);                // Depth Buffer Setup
    glEnable(GL_DEPTH_TEST);           // Validare test adancime
    glDepthFunc(GL_LEQUAL);           // Tipul testului
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST); //Calcul proiectie
//perspectiva
    return TRUE;          // Initializare OK
}

//desenare efectiva scena
void CALLBACK DeseneazaScena()
{

```

```

// Stergere ecran si buffer de adancime
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity(); // Reset matrice ModelView curenta

glPushMatrix(); // salvare matrice de lucru pe stiva
glTranslatef(-2.0f, 0.0f, -6.0f); // plasare triunghi stang
glTranslatef(0.0f, fTranslate, 0.0f); // translatie pe Y
Draw_Triangle(); // desenare triunghi
glPopMatrix(); // restaurare matrice de lucru din varful stivei

glPushMatrix();
glTranslatef(0.0f, 0.0f, -6.0f); // plasare triunghi central
glRotatef(fRotate, 0, 1.0f, 0); // rotire in jurul axei Y
Draw_Triangle(); // desenare triunghi
glPopMatrix();

glPushMatrix();
glTranslatef(2.0f, 0.0f, -6.0f); // plasare triunghi drept
glScalef(fScale, fScale, fScale); // scalare cu aceeasi valoare pe
// toate directiile (x,y,z)
Draw_Triangle(); // desenare triunghi
glPopMatrix();

auxSwapBuffers();
}

void Draw_Triangle() // desenare tringhi
{
    glBegin(GL_TRIANGLES);
        glColor3f(1.0f, 0.0f, 0.0f);
        glVertex3f(0.0f, 1.0f, 0.0f);
        glColor3f(0.0f, 1.0f, 0.0f);
        glVertex3f(-1.0f, -1.0f, 0.0f);
        glColor3f(0.0f, 0.0f, 1.0f);
        glVertex3f( 1.0f, -1.0f, 0.0f);
    glEnd();
}

void CALLBACK invarte() // modificare variabile pt. rot., transl., scalare
{
    fTranslate += 0.005f;
    fRotate    += 0.5f;
    fScale     -= 0.005f;

    if(fTranslate > 0.9f) fTranslate = 0.0f;
    if(fScale < 0.1f)    fScale    = 1.0f;
    DeseneazaScena();
}

void CL2_1Dlg::OnBnClickedButton1()
{
    // Stabilirea ferestrei de lucru OpenGL
    auxInitDisplayMode(AUX_DOUBLE|AUX_RGBA);
    auxInitPosition(10, 10, 500, 500);
    auxInitWindow("Translatii, rotatii, scalari");

    InitGL();
    // Înregistrarea functiilor CALLBACK
    auxReshapeFunc(ModificaDimensiune);
    auxIdleFunc(invarte);
}

```

```

    auxMainLoop (DeseneazaScena) ;
}

```

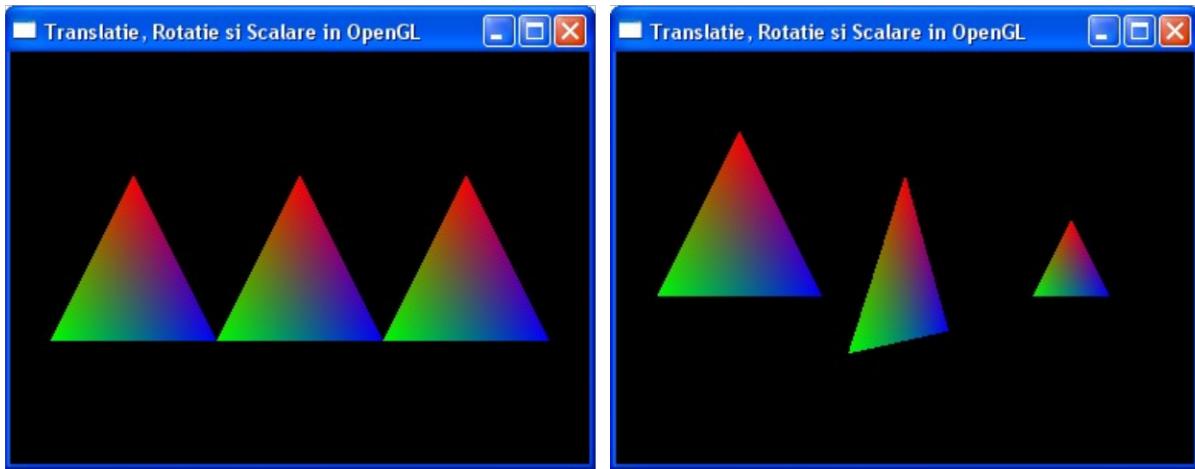


Figura 8 – Manipulare obiecte (triunghiuri) in OpenGL folosind translatii, rotatii si scalarii

Întrebări:

Exerciții:

1. Deseneați un lanț de triunghiuri folosind primitivele GL_TRIANGLE_STRIP (Figura aaa) și GL_TRIANGLE_FAN (Figura aab)

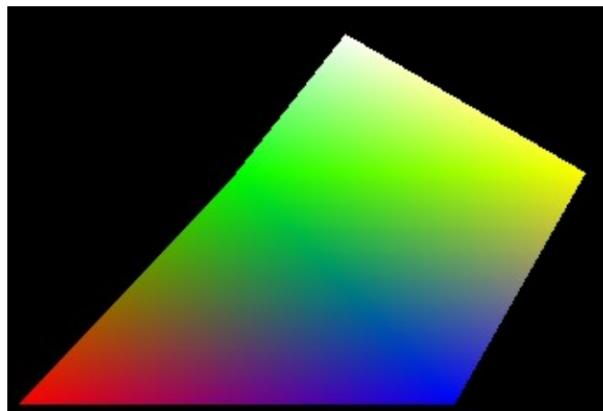


Figura aaa GL_TRIANGLE_STRIP

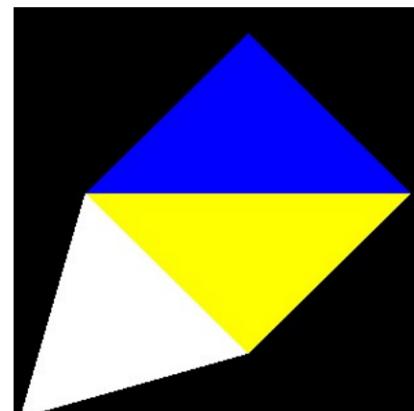


Figura aab GL_TRIANGLE_FAN

2. Să se deseneze în OpenGL o sferă de culoare galbenă (se folosește funcția auxSolidSphere ()) care să se deplaseze, la apăsarea tastei UP, pe axa y de la -50 la +50.
- 3.

OpenGL superbile
www.opengl.org