Laborator 7
Transformări ale obiectelor 3D în OpenGL

7.1. Etapele transformării din spaţiul 3D pe suprafaţa de afişare

OpenGL utilizează un sistem de coordonate 3D dreapta.

Secvenţa de transformări aplicate asupra punctelor prin care este definit un obiect 3D pentru a fi afişat pe ecran este următoarea (figura II.8):

- transformarea de modelare şi vizualizare (ModelView)
- transformarea de proiecţie, însoţită de decupare la marginile volumului vizual canonic
- împărţirea perspectivă
- transformarea în poarta de vizualizare din fereastra curentă

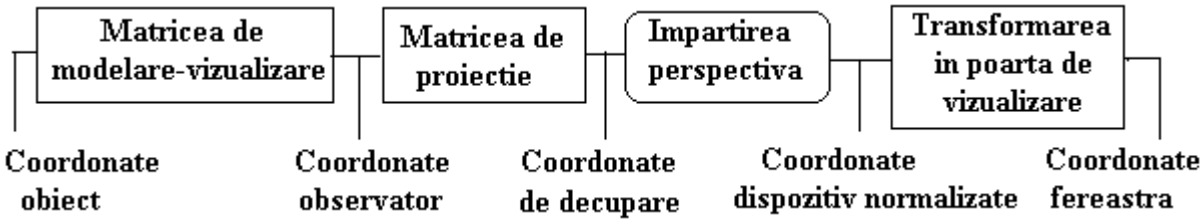


Figura 7.1 Secvenţa de transformari aplicate vârfurilor în OpenGL

Procesul de transformări necesar producerii imaginii dorite pentru a fi redată pe o suprafaţă de afişare este asemănator cu cel al efectuării unei fotografii. Aceşti paşi ar fi următorii:

- Aranjarea scenei pentru a fi fotografiată în contextul dorit - **transformarea de modelare**;
- Aranjarea aparatului de fotografiat şi încadrarea scenei -**transformarea de vizualizare**;
- Alegerea lentilelor aparatului sau modificarea zoom-ului - **transformarea de proiecţie**;
- Determinarea dimensiunii imaginii finale - **transformarea în poarta de vizualizare**.

7.2. Transformarea de modelare şi vizualizare

O aceeaşi imagine a unui obiect se poate obţine în două feluri :

- poziţionând obiectul în faţa unei camere de luat vederi fixe
- poziţionând camera de luat vederi în faţa obiectului fix

Prima operaţie corespunde unei transformări de modelare a obiectului. Cea de a doua operaţie corespunde unei transformări de vizualizare. Deoarece ambele pot fi folosite pentru a obţine o aceeaşi imagine, sunt tratate împreună, ca o singură transformare

Transformarea de modelare are drept scop poziţionarea obiectelor în scena 3D. Această transformare este necesară deoarece, în mod uzual, fiecare obiect este definit ca *obiect unitate* într-un *sistem de coordonate local*. De exemplu, un cub poate fi definit ca având latura de o unitate, centrat în originea unui sistem de coordonate carteziene 3D. Reprezentarea la mărimea dorită, poziţionarea şi orientarea sa în scena 3D, care este definită într-un sistem de coordonate *global*, poate să presupună o transformare compusă din scalare şi rotaţie faţă de originea sistemului de coordonate local, urmată de o translaţie. Prin această transformare se crează o *instanţă a cubului*, de aceea transformarea de modelare se mai numeşte şi *transformare de instanţiere*.

Transformarea de modelare este o transformare compusă din transformări geometrice simple care poate fi definită ca produs matricial, folosind funcţiile de translaţie, rotaţie şi scalare oferite de OpenGL.

Transformarea de vizualizare este determinată de poziţia observatorului (camera de luat vederi), direcţia în care priveşte acesta si direcţia sus a planului de vizualizare. In mod implicit, observatorul este situat in originea sistemului de coordonate în care este descrisă scena 3D, direcţia în care priveşte este direcţia negativa al axei OZ, iar direcţia sus a planului de vizualizare este direcţia pozitiva a axei OY. Cu aceste valori implicite, transformarea de vizualizare este transformarea identică.

Funcţia **gluLookAt** permite modificarea parametrilor implicişi ai transformării de vizualizare :

```
void gluLookAt(GLdouble eyex,GLdouble eyey,GLdouble eyez,
  GLdouble centerx,GLdouble centery,GLdouble centerz,
  GLdouble upx,GLdouble upy,GLdouble upz);
```

- *eyex, eyey, eyez* reprezintă poziţia observatorului
- *centerx, centery, centerz* reprezintă direcţia în care se priveşte
- *upx, upy, upz* reprezintă direcţia vectorului « sus » al planului de vizualizare

Poziţia observatorului reprezintă punctul de referinţă al vederii, R, iar direcţia în care se priveşte este direcţia normalei la planul de vizualizare, în R. Vectorul « sus » determină direcţia pozitivă a axei verticale a sistemului de coordonate 2D ataşat planului de vizualizare. Sistemul de coordonate 2D ataşat planului de vizualizare împreună cu normala la plan formează sistemul de coordonate 3D al planului de vizualizare, care în terminologia OpenGL este numit **sistemul de coordonate observator**.

Funcţia **gluLookAt** construieşte **matricea transformării din sistemul de coordonate obiect în sistemul de coordonate observator** şi o înmulţeşte la dreapta cu matricea curentă.

In OpenGL transformarea de modelare şi de vizualizare sunt exprimate printr-o singură matrice de transformare, care se obţine înmulţind matricile celor două

transformări. Ordinea de înmulţire a celor două matrici trebuie să corespundă ordinei în care ar trebui aplicate cele două transformări : mai întâi transformarea de modelare apoi transformarea de vizualizare.

In OpenGL punctele 3D se reprezintă prin vectori coloană. Astfel, un punct (x,y,z) se reprezintă în coordonate omogene prin vectorul $[x_w\ y_w\ z_w\ w]^T$. Daca A, B şi C sunt 3 matrici de transformare care exprimă transformările de aplicat punctului în ordinea A, B, C, atunci secvenţa de transformări se exprimă matricial astfel :

$[x_{wô}\ y_{wô}\ z_{wô}\ wô]^T = C \times B \times A \times [x_w\ y_w\ z_w\ w]^T$

Transformarea de modelare şi vizualizare este o transformare compusă, reprezentată printr-o matrice VM, ce se obţine înmulţind matricea transformării de vizualizare cu matricea transformării de modelare. Fie V şi M aceste matrici. Atunci,

VM = V × M

Dacă coordonatele unui vârf în sistemul de coordonate obiect sunt reprezentate prin vectorul $[xo\ yo\ zo\ wo]^T$, atunci coordonatele vârfului în sistemul de coordonate observator (ñeye coordinatesò) se obţin astfel:

$$[xe\ ye\ ze\ we]^T = VM \times [xo\ yo\ zo\ wo]^T$$

Matricea VM este aplicată automat şi vectorilor normali.

7.3. Transformarea de proiecţie

Matricea de proiecţie este calculată în OpenGL în funcţie de tipul de proiecţie specificat de programator şi parametrii care definesc volumul de vizualizare. Matricea de proiecţie este matricea care transformă volumul vizual definit de programator într-un volum vizual canonic. Această transformare este aplicată vârfurilor care definesc primitivele geometrice în coordonate observator, rezultând **coordonate normalizate.** Primitivele reprezentate prin coordonate normalizate sunt apoi decupate la marginile volumului vizual canonic, de aceea coordonatele normalizate se mai numesc şi **coordonate de decupare**. Volumul vizual canonic este un cub cu latura de 2 unităţi, centrat în originea sistemului coordonatelor de decupare. După aplicarea transformării de proiecţie, orice punct 3D (din volumul vizual canonic) se proiectează în fereastra 2D printr-o proiecţie ortografică (xô =x, yô=y) condiţie necesară pentru aplicarea algoritmului z-buffer la producerea imaginii.

Dacă coordonatele unui vârf în sistemul de coordonate observator sunt reprezentate prin vectorul $[xe\ ye\ ze\ we]^T$, iar P este matricea de proiecţie, atunci coordonatele de decupare ale vârfului (ñclip coordinatesò) se obţin astfel:

$$[xc\ \ yc\ \ zc\ wc]^T = P \times [xo\ \ yo\ \ zo\ \ wo]^T$$

Prezentăm în continuare funcţiile OpenGL prin care pot fi definite proiecţiile şi volumul de vizualizare.

```
void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble
top, GLdouble near, GLdouble far);
```

Funcţia defineşte o proiecţie perspectivă cu centrul de proiecţie în poziţia observatorului (punctul de referinţă al vederii). Volumul de vizualizare (figura II.9) este delimitat prin « planul din faţă » şi « planul din spate », plane paralele cu planul de vizualizare, definite prin distanţele lor faţă de poziţia observatorului (planul de vizualizare). Planul din faţă va fi folosit ca plan de proiecţie. Lui i se ataşează un sistem de coordonate 2D având axa verticală (sus) orientată ca şi axa verticală a planului de vizualizare. Deschiderea camerei de luat vederi este determinată printr-o fereastră rectangulară, cu laturile paralele cu axele, definită în planul de proiecţie.

- *(left, bottom)* şi *(right, top)* reprezintă colţurile ferestrei din planul din faţă
- *znear, zfar* reprezintă distanţele de la poziţia observatorului la planul din faţă, respectiv spate. Ambele distanţe trebuie să fie pozitive.

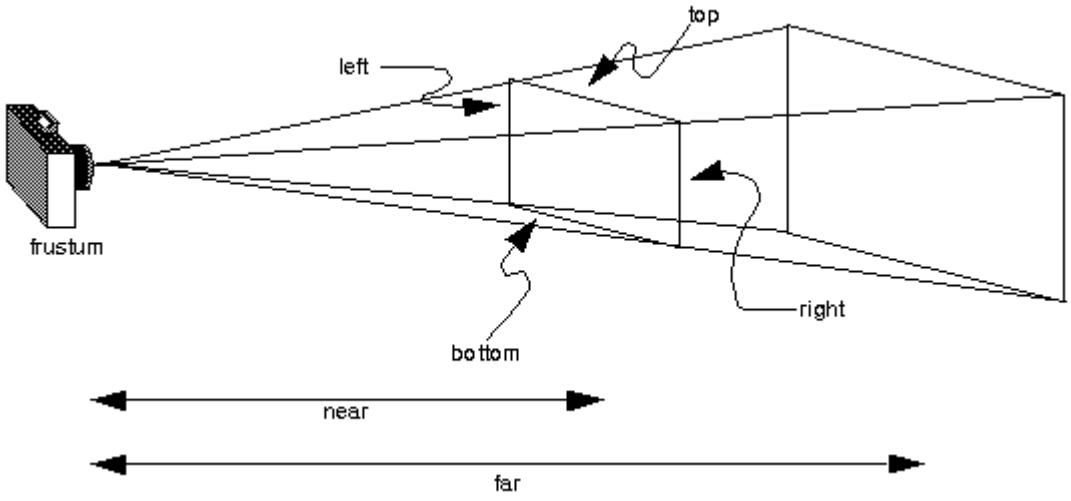


Figura 7.2. Volumul vizual perspectivă

Dacă *left=right* sau *bottom=top* sau *znear=zfar* sau *znear*<=0 sau *zfar*<=0 se semnalează eroare. Colţurile ferestrei 2D din planul de proiecţie, (left, bottom,-near) şi (right, top,-near) sunt mapate pe colţurile stânga-jos şi dreapta-sus ale ferestrei 2D din sistemul coordonatelor de decupare, adică (-1,-1,-1) şi (1,1,-1). Matricea de proiecţie este în acest caz :

$$P = \begin{bmatrix} \frac{2*near}{right - left} & 0 & A & 0 \\ 0 & \frac{2*near}{top - bottom} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

unde

$$A = \frac{right + left}{right - left} \quad , \quad B = \frac{top + bottom}{top - bottom}$$

$$C = -\frac{far + near}{far - near} \quad , \quad D = -\frac{2 * far * near}{far - near}$$

Funcţia **glFrustum** înmulţeşte matricea curentă cu matricea de proiecţie rezultată şi memorează rezultatul în matricea curentă. Astfel, dacă M este matricea curentă şi P este matricea proiecţiei, atunci **glFrustum** înlocuieşte matricea M cu M * P.

O altă funcţie care poate fi folosită pentru proiecţia perspectivă este **gluPerspective :**

```
void gluPerspective( GLdouble fovy, GLdouble aspect, GLdouble near,
GLdouble far);
```

Funcţia **gluPerspective** creează un volum de vizualizare la fel ca şi funcţia **glFrustum**, dar în acest caz el este specificat în alt mod. În cazul funcţiei **gluPerspective** acesta se specifică prin unghiul de vizualizare în planul XOZ (deschiderea camerei de luat vederi) şi raportul dintre lăţimea şi înălţimea ferestrei definite în planul de aproape (pentru o fereastră plătită acest raport este 1.0.)

- *fovy* reprezintă unghiul de vizualizare în planul XOZ, care trebuie să fie în intervalul [0.0,180.0].
- *aspect* reprezintă raportul lăţime / înălţime al laturilor ferestrei din planul de aproape; acest raport trebuie să corespundă raportului lăţime/înălţime asociat porţii de afişare. De exemplu, dacă *aspect* = 2.0 atunci unghiul de vizualizare este de două ori mai larg pe direcţia x decât pe y. Dacă poarta de afişare este de două ori mai lată decât înălţimea atunci imaginea va fi afişată nedistorsionată.
- *near* şi *far* reprezintă distanţele între observator şi planele de decupare de-a lungul axei z negative. Întotdeauna trebuie să fie pozitivi.

Proiecţia ortografică este specificată cu ajutorul funcţiei **glOrtho :**

```
void glOrtho( GLdouble left, GLdouble right, GLdouble bottom, GLdouble
top, GLdouble near, GLdouble far);
```

unde

- *(left, bottom)* şi *(right, top)* reprezintă colţurile ferestrei din planul din faţă
- *near, far* reprezintă distanţele de la poziţia observatorului la planul din faţă, respectiv planul din spate

Volumul de vizualizare este în acest caz un paralelipiped dreptunghic delimitat de planul din faţă şi cel din spate, plane paralele cu planul de vizualizare (perpendiculare pe axa z), precum şi de planele sus, jos, dreapta , stânga. Direcţia de proiecţie este dată de axa Z a sistemului de coordonate atașat planului de vizualizare.

Fereastra definită în planul din faţă, (left, bottom,-near) şi (right, top,-near), este mapată pe fereastra 2D din sistemul coordonatelor de decupare (-1,-1,-1) şi (1,1,-1). Matricea de proiecţie este în acest caz :

$$P = \begin{bmatrix} \frac{2}{right - left} & 0 & 0 & t_x \\ 0 & \frac{2}{top - bottom} & 0 & t_y \\ 0 & 0 & \frac{-2}{far - near} & t_z \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Unde :

$$t_x = -\frac{right + left}{right - left},\ t_y = -\frac{top + bottom}{top - bottom},\ t_z = -\frac{far + near}{far - near}$$

Matricea curentă este înmulţită cu matricea de proiecţie rezultată, rezultatul fiind depus în matricea curentă.

Tot pentru proiecţia ortografică poate fi folosită şi funcţia **gluOrtho2D** care defineşte matricea de proiecţie ortografică în care near=-1 şi far=1.

```
void gluOrtho2D( Gldouble left, Gldouble right,
           Gldouble bottom, Gldouble top);
```

unde

- *(left, bottom)* şi *(right, top)* reprezintă colţurile ferestrei din planul din faţă

7.4. Impartirea pespectivă

Prin această operaţie se obţin coordonate 3D în spaţiul coordonatelor de decupare, pornind de la coordonatele 3D omogene :

xd = xc/wc , yd = yc/wc , zd= zc/wc

Coordonatele (xd, yd, zd) sunt numite « coordonate dispozitiv normalizate ».
Operaţia este denumită « împărţire perspectivă » deoarece numai în cazul unei proiecţii perspectivă wc este diferit de 1.

7.5. Transformarea în poarta de vizualizare

Această transformare se aplică coordonatelor dispozitiv normalizate pentru a se obţine coordonate raportate la sistemul de coordonate al fereastrei curente de afişare. Este o transformare fereastră-poartă, în care fereastra este fereastra 2D din sistemul coordonatelor normalizate, având colţurile în (-1, -1, -1)-(1, 1, -1), iar poarta este un dreptunghi din fereastra de afişare care poate fi definit prin apelul funcţiei **glViewport**.

```
void glViewport(GLint px, GLint Py, GLsizei width, GLsizei height );
```

- (*px, py*) reprezintă coordonatele în fereastră ale colţului stânga jos al porţi de afişare (în pixeli). Valorile implicite sunt (0,0).
- *width, height* reprezintă lăţimea, respectiv înălţimea porţi de afişare. Valorile implicite sunt date de lăţimea şi înălţimea ferestrei curente de afişare.

Fie (xd, yd, zd) coordonatele dispozitiv normalizate ale unui vârf şi (xw, yw, zw) coordonatele vârfului în fereastra de afişare. Transformarea în poarta de vizualizare este definită astfel :

xw = ox + (width/2)xd
yw = oy + (height/2)yd
zw = ((f-n)/2)zd + (n+f)/2

unde (ox, oy) reprezintă coordonatele centrului porţiunii de afişare.
n şi f au valorile implicite 0.0 respectiv 1.0, dar pot fi modificate la valori cuprinse în intervalul [0,1] folosind funcţia **DepthRange**. Dacă n şi f au valorile implicite, atunci prin transformarea în poarta de vizualizare volumul vizual canonic se transformă în cubul cu colţurile de minim şi maxim în punctele (px, py, 0)-(px+width, py+height,1).
Calculul coordonatei zw este necesară pentru comparaţiile de adâncime în algoritmul z-buffer.
Observaţie: în programul din fişierul exemplu5.c lăţimea şi înălţimea porţiunii de afişare sunt specificate folosind lăţimea şi înălţimea ferestrei curente a aplicaţiei.

7.6. Funcţii de operare cu matrici de transformare

Transformarea de modelare şi vizualizare şi transformarea de proiecţie sunt reprezentate prin matrici. Programatorul poate încărca una dintre aceste matrici cu o matrice proprie sau cu matricea unitate, sau o poate înmulţi cu alte matrici.

Înaintea specificării unei transformări trebuie selectată **matricea curentă**, care va fi modificată. Pentru aceasta se va apela funcţia **glMatrixMode**.

```
void glMatrixMode( GLenum mode );
```

Parametrul *mode* reprezintă matricea asupra căreia se vor efectua modificările ulterioare. El poate avea una dintre următoarele valori:

- **GL_MODELVIEW** – matricea curentă va fi matriciea de modelare şi vizualizare
- **GL_PROJECTION** – matricea curentă va fi matricea de proiecţie.
- **GL_TEXTURE** – matricea curentă va fi matricea textură.

In programul din fişierul exemplu5.c, înaintea transformării de vizualizare, matricea curentă este setată la matricea identitate (de 4 linii şi 4 coloane) cu ajutorul functiei **glLoadIdentity**.

```
void glLoadIdentity(void);
```

Apelul funcţiei **glLoadIdentity** este necesar deoarece majoritatea funcţilor de transformare înmulţesc matricea curentă cu matricea specificată şi rezultatul este depus în matricea curentă. Dacă matricea curentă nu este setată iniţial la matricea identitate, atunci se vor folosi matricile de transformare anterioare combinate cu cea furnizată în acel moment

Dacă se doreşte ca o anumită matrice să devină matricea curentă, atunci se va folosi funcţia **glLoadMatrix***.

```
void glLoadMatrix{fd}(const TYPE *m);
```

Valorile conţinute în vectorul *m* se memorează în matricea curentă, în ordinea coloanelor :

$$\mathbf{M} = \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

Observaţie : Elementele unei matrici din limbajul C sunt memorate în ordinea liniilor.

Funcţia **glMultMatrix*** se poate folosi pentru a înmulţi matricea curentă cu matricea dată ca parametru funcţiei **glMultMatrix***.

```
void glMultMatrix{fd}(const TYPE *m);
```

Funcţia înmulţeşte matricea specificată de parametrul *m* cu matricea curentă şi rezultatul înmulţirii este stocat în matricea curentă. Dacă C este matricea curentă, atunci rezultatul înmulţirii este C=CÅm.

Pentru specificarea unei translaţii se poate folosi funcţia:

```
void glTranslatef(GLfloat x, GLfloat y, GLfloat z);
```

unde *x*, *y* şi *z* reprezintă componentele vectorului de translaţie

Matricea curentă este înmulţită cu matricea de translaţie şi rezultatul înlocuieşte matricea curentă. Dacă M este matricea curentă şi T este matricea de translaţie, atunci M este înlocuită cu MÅT.

Daca matricea curenta este GL_MODELVIEW sau GL_PROJECTION, toate obiectele afişate după apelul functiei **glTranslatef** vor fi translatate.

Pentru rotaţii se poate folosi funcţia:

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

- *angle* reprezintă unghiul de rotaţie exprimat în grade.
- *x*, *y*, *z* reprezintă coeficienţii directori ai axei în jurul căreia se realizează rotaţia.

Funcţia efectuează o rotaţie în sens trigonometric cu unghiul specificat, în jurul vectorului ce uneşte originea cu punctul de coordonate (x, y, z).
Matricea curentă este înmulţită cu matricea de rotaţie şi rezultatul înlocuieşte matricea curentă. Dacă M este matricea curentă şi R este matricea de rotaţie, atunci M este înlocuită cu MÅR.
Dacă matricea curentă este **GL_MODELVIEW** sau **GL_PROJECTION**, toate obiectele afişate după apelul funcţiei **glRotatef** vor fi rotite.

Pentru scalare se poate folosi funcţia:

```
void glScalef(GLfloat x, GLfloat y, GLfloat z);
```

unde x, *y*, *z* reprezintă factorii de scalare de-a lungul axelor x, y şi z.
Matricea curentă este înmulţită cu matricea de scalare şi rezultatul înlocuieşte matricea curentă. Dacă M este matricea curentă şi S este matricea de translaţie, atunci M este înlocuită cu M*S.

Dacă matricea curentă este **GL_MODELVIEW** sau **GL_PROJECTION**, atunci toate obiectele afişate după apelul funcţiei **glScalef** vor fi scalate.
Folosirea funcţiei **glScale*** diminuează performanţele calculului iluminării, deoarece vectorii normală trebuie renormalizaţi după această transformare.

Exemplu
Considerăm următoarea secvenţă care specifică o secvenţă de trei transformări:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(N);            /* aplică transformarea N */
glMultMatrixf(M);            /* aplică transformarea M */
glMultMatrixf(L);            /* aplică transformarea L */
glBegin(GL_POINTS);
  glVertex3f(v);             /* afişează vârful v transformat*/
glEnd();
```

La execuţia secvenţei de mai sus, matricea de modelare şi vizualizare (MODELVIEW) va conţine succesiv : **I**, **N**, **N•M**, şi în final **N•M•L**, unde **I** reprezintă matricea identitate. Transformarea aplicată vârfului **v** va fi **NML•v**, unde **NML = N•M•L**.

7.7. Gestiunea stivelor de matrici

Un sistem OpenGL păstrează câte o stivă de matrici pentru fiecare mod matrice selectat cu ajutorul funcţiei **MatrixMode**. Astfel, există :

- Stiva matricilor Modelview (cel puţin 32 matrici 4 x 4);
- Stiva matricilor de proiecţie (cel puţin 2 matrici 4 x 4) ;
- Stiva matricilor textură(cel puţin 2 matrici 4 x 4).

Matricea curentă este întotdeauna matricea din vârful stivei corespunzătoare modului matrice curent.
O stivă de matrici este folositoare pentru construirea modelelor ierarhice în care sunt construite obiecte complexe pornind de la obiecte simple. De exemplu, să presupunem că se desenează o maşină şi există o singură rutină care desenează o roată. Această rutină desenează roata într-o anumită poziţie şi orientare. Când se desenează maşina, rutina de afişare a roţii se va apela de 4

ori, aplicându-se diferite transformări pentru a poziţiona corect roţile. De exemplu, pentru desenarea primei roţi aceasta trebuie să fie translatată. La desenarea celei de a doua roţi trebuie aplicată o altă translaţie (dar faţă de poziţia iniţială ï nu trebuie ţinut cont de prima translaţie) şi în acelaşi mod pentru desenarea celorlalte roţi.

Deoarece transformările sunt păstrate ca matrici, o stivă de matrici furnizează un mecanism util pentru efectuarea unei transformări ca apoi să se realizeze o altă transformare fără a se mai ţine cont de transformarea anterioară. Toate operaţiile cu matrici (**glLoadMatrix**, **glMultMatrix**, **glLoadIdentity** şi funcţiile care creează matrici de transformare specifice) lucrează cu matricea curentă sau cu matricea din vârful stivei.

Pentru lucrul cu stivele de matrici OpenGL pune la dispoziţie funcţiile **glPushMatrix** şi **glPopMatrix**.

```
void glPushMatrix(void);
```

Funcţia adaugă un nou element la stiva curentă şi memorează matricea curentă atât în elementul din vârful stivei cât şi în intrarea următoare. Stiva curentă este determinată de ultimul apel al funcţiei **glMatrixMode**. Dacă prin adăugare se depăşeşte capacitatea stivei se generează eroare.

```
void glPopMatrix(void);
```

Funcţia elimină intrarea din vârful stivei şi înlocuieşte matricea curentă cu matricea care era memorată în a 2-a intrare a stivei. Dacă stiva avea o singură intrare, apelul funcţiei **glPopMatrix** generează eroare.

Exemplu

Programul din fişierul exemplu5.c afişează un cub care este mai întâi scalat (transformarea de modelare). Transformarea de vizualizare constă dintr-o translaţie a poziţiei observatorului pe axa z, în poziţia (0,0,5). Observatorul priveşte spre origine iar direcţia axei sus este direcţia axei OY a sistemului de coordonate obiect.

```
// cub.cc
#include <GL\glut.h>

void init(void)
{   glClearColor (0.0, 0.0, 0.0, 0.0);
}

void display(void)
{
        // sterge fereastra.
       glClear(GL_COLOR_BUFFER_BIT);

       glLoadIdentity ();
   gluLookAt(0.0,0.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
```

```
   glScalef (1.0, 2.0, 1.0);
   glutWireCube (1.0);
   glFlush ();
}
void reshape (int w, int h)
{  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   glFrustum (-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
   glMatrixMode (GL_MODELVIEW);
}
int main(int argc, char** argv)
{  glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (500, 500);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutMainLoop();
   return 0;
}
```

Transformarea de vizualizare ki modelare este creatŁ în funcŞia **display** unde este apelatŁ ki funcŞia de afikare a cubului, **glutWireCube**. Astfel funcŞia **display** poate fi folositŁ în mod repetat pentru a afika conŞinutul ferestrei (de exemplu în cazul în care fereastra este mutatŁ pe ecran).

Transformarea de proiecŞie ki transformarea în poarta de vizualizare sunt specificate în funcŞia **Reshape**, care este apelatŁ de sistem ori de câte ori este redimensionatŁ fereastra aplicaŞiei.

Efectul obŞinut prin mutarea observatorului în spatele cubului (folosind transformarea de vizualizare) se poate obŞine ki prin deplasarea cubului, folosind o transformare de modelare.

Exerciţii :

1. Să se execute următorul program să se modificei fereastra inițială mărimea și / sau poziția, și urmăriți efectele.
2. Înlocuiți cu reprezentarea (wireframe) – glutWireTeapot- în loc de reprezentarea colorată `glutSolidTeapot`, şi modificați dimensiunea ceainicului

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

void display(void)
{
    // sterge fereastra.
   glClear(GL_COLOR_BUFFER_BIT);
    // deseneaza scena.
     glutSolidTeapot(.5);
    // Golirea tamponului de afisare

    glFlush();
    // este apelat dublu buffer
   // glutSwapBuffers();
}

int main (int argc, char * argv[] )
{
   // Initializare GLUT.
    glutInit(&argc,argv);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutInitDisplayMode(GLUT_RGB);
   glutCreateWindow("Ceainic!");
    glutDisplayFunc(display);
   glutMainLoop();

}
```

3.Introduceţi în programul de mai sus transformarea în poartŁ de vizualizare, aка cum este prezentat mai jos.

```
// Salut ceainic.cc
#include <GL\glut.h>


void display(void)
{
        // sterge fereastra.
       glClear(GL_COLOR_BUFFER_BIT);
    // deseneaza scena.
     glutSolidTeapot(.5);
    // Golirea tamponului de afisare

    glFlush();
    // este apelat dublu buffer
   // glutSwapBuffers();
}
       void reshape ( int width, int height )
{
              // Defineste transformarea in poarta de afisare
              glViewport(0,0,width,height);

}

int main (int argc, char * argv[] )
{
```

```
      // Initializare GLUT.
    glutInit(&argc,argv);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutInitDisplayMode(GLUT_RGB);
      glutCreateWindow("Ceainic!");
    glutDisplayFunc(display);
      glutReshapeFunc(reshape);
      glutMainLoop();

}
```

4.ÎncŁrcaŭi programele de mai jos.

```
// ceainic3.cc
#include <GL\glut.h>


void display(void)
{
       // sterge fereastra.
      glClear(GL_COLOR_BUFFER_BIT);

      // Matricea de vizualizare ModelView va
       // afecta modificarile viitoarei matrici
      glMatrixMode(GL_MODELVIEW);

     // deseneaza scena.
      glutSolidTeapot(0.5);
     // Golirea tamponului de afisare

     glFlush();
     // este apelat dublu buffer
    // glutSwapBuffers();
}

void reshape ( int width, int height ) {

     /* define the viewport transformation */
     glViewport(0,0,width,height);

}


int main (int argc, char * argv[] )
{
      // Initializare GLUT.
     glutInit(&argc,argv);
      // Seteaza dimensiunea.pozitia si
     //modul de afisare pentru noua fereastra
     glutInitWindowSize(500,500);
     glutInitWindowPosition(0,0);
     glutInitDisplayMode(GLUT_RGB);
      // Noua fereastra va fi creata
      glutCreateWindow("Ceainic!");
     glutDisplayFunc(display);
      glutReshapeFunc(reshape);
      // Transformarea de proiectie este definita
      glMatrixMode(GL_PROJECTION);
     glLoadIdentity();
     glOrtho(-2.0,2.0,-2.0,2.0,-2.0,2.0);
      // Comunica GLUT  desfasurarea evenimente
      glutMainLoop();

}
```

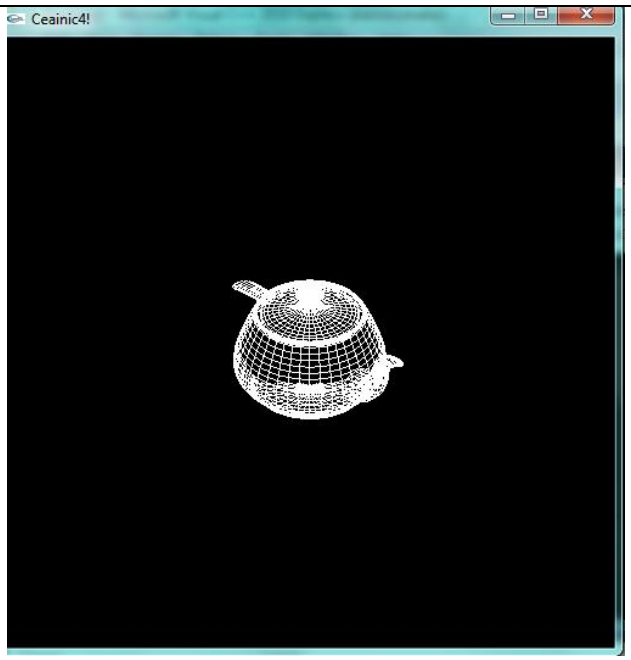
DacŁ rula”Hacest program, ar trebui sŁ observa”H un lucru nou: ceainicul apare mai la o dimensiune mult mai micŁ decât cel precedent.

ExistŁ douŁ linii de cod noi care definesc în primul rând, transformarea de proiec”Hie:

```
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
glOrtho (-2.0,2.0, -2.0,2.0, -2.0,2.0);
```

Aceste linii principale, ar fi trebuit sa fie definite înainte de apari”Hia oricŁrui desen.
glMatrixMode precizeazŁ care dintre matricele de vizualizare trebuie sŁ fie modificate
-GL_PROJECTION precizeazŁ cŁ ulterioarele comenzile vor afecta stiva de transformare de proiec”Hie. Ini”Hial, existŁ o singurŁ matrice de un fel pe stiva de proiec”Hie:
glOrtho specificŁ proiec”Hia paralelŁ ortogonalŁ:

```
// ceainic3.cc
#include <GL\glut.h>

void display(void)
{
       // sterge fereastra.
      glClear(GL_COLOR_BUFFER_BIT);
      // Matricea de vizualizare ModelView va
        //afecta modificarile viitoarei matrici
      glMatrixMode(GL_MODELVIEW);
     // deseneaza scena.
      glutWireTeapot(0.5);
     // Golirea tamponului de afisare
          glFlush();
    // este apelat dublu buffer
   // glutSwapBuffers();
}

void reshape ( int width, int height ) {
    /* define the viewport transformation */
    glViewport(0,0,width,height);
}
int main (int argc, char * argv[] )
{
      // Initializare GLUT.
    glutInit(&argc,argv);
      // Seteaza dimensiunea.pozitia si
//modul de afisare pentru noua fereastra
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutInitDisplayMode(GLUT_RGB);
      // Noua fereastra va fi creata
      glutCreateWindow("Ceainic4!");
    glutDisplayFunc(display);
      glutReshapeFunc(reshape);
      // Transformarea de proiectie este
definita
      glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2.0,2.0,-2.0,2.0,-2.0,2.0);
      // Defineste pozitia de vizualizare
      glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

gluLookAt(1.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0);
      // Comunica GLUT  desfasurarea evenimente
      glutMainLoop();

}
```



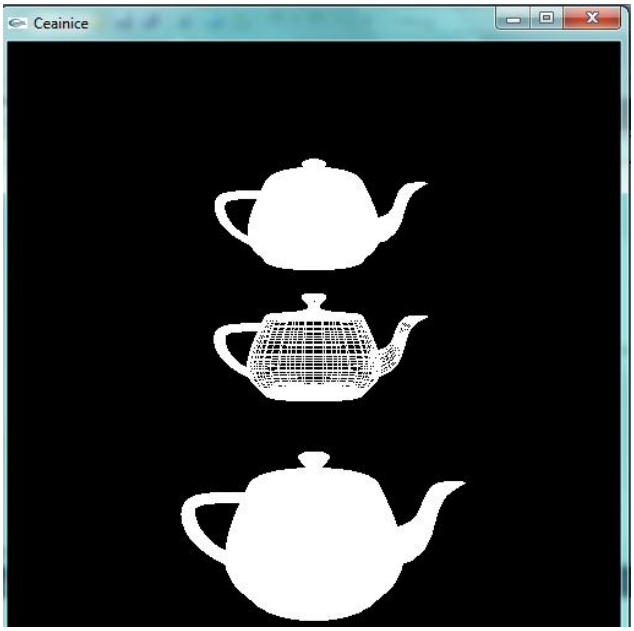
1. Încercaţi sŁ modificaţi parametrii `gluLookAt`. Ce se va intâmpla кi de ce?

5.Aplicati transformŁri de modelare pe programul de mai jos.

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

```
// ceainic3.cc
#include <GL\glut.h>

void display(void)
{
        // sterge fereastra.
       glClear(GL_COLOR_BUFFER_BIT);
       // Matricea de vizualizare ModelView va afecta modificarile viitoarei matrici
       glMatrixMode(GL_MODELVIEW);
    // deseneaza scena.
       glPushMatrix();
    glPushMatrix();
       glTranslatef(0,0,-3);
     glutWireTeapot(1);  // Ceainic mijlociu
        glTranslatef(0,2,0);
        glutSolidTeapot(1);  // Ceainicul de sus
        glPopMatrix();
        glTranslatef(0,-2,-1);
    glutSolidTeapot(1);      // Ceainicul de jos

       glPopMatrix();
    // Golirea tamponului de afisare

    glFlush();
    // este apelat dublu buffer
   // glutSwapBuffers();
}

void reshape ( int width, int height ) {

    /* define the viewport transformation */
    glViewport(0,0,width,height);

}
int main (int argc, char * argv[] )
{
       // Initializare GLUT.
    glutInit(&argc,argv);
       // Seteaza dimensiunea.pozitia si modul de afisare pentru noua fereastra
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutInitDisplayMode(GLUT_RGB);
       // Noua fereastra va fi creata
       glutCreateWindow("Ceainice");
    glutDisplayFunc(display);
       glutReshapeFunc(reshape);
       // Transformarea de proiectie este definita
       glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60,1,1,10);
       // Defineste pozitia de vizualizare
       glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0.0,0.0,5.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0);

       // Comunica GLUT  desfasurarea evenimente
       glutMainLoop();
```

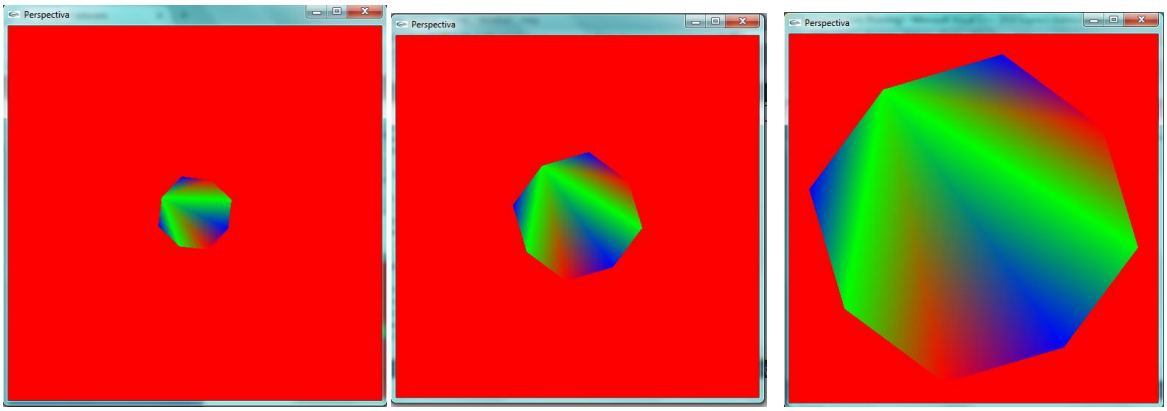
}



6.Rulaţi programul de mai jos pentru transformarea de modelare si de vizualizare.

```
// - Vizualizarea Volumului Proiectiei Perspectiva
// - Incearca si apelarea tastaturii
// - Functia callback de reconfigurare
// - Temporizator
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <GL/glut.h>
float z_pos=0.0f;
float rot=0.0f;
void mydisplay(){
glClear(GL_COLOR_BUFFER_BIT);
glLoadIdentity();
glTranslatef(0.0,0.0f,z_pos);
glRotatef(rot, 0, 0, 1);
glBegin(GL_POLYGON);
glColor3f(0, 1, 0);
glVertex3f(-0.5, -0.5, -5);
glColor3f(0, 0, 1);
glVertex3f(-0.75, 0, -5);
glColor3f(1, 0, 0);
glVertex3f(-0.5, 0.5, -5);
glColor3f(0, 1, 0);
glVertex3f(0, 0.75, -5);
glColor3f(0, 0, 1);
glVertex3f(0.5, 0.5, -5);
```

```
glColor3f(1, 0, 0);
glVertex3f(0.75, 0, -5);
glColor3f(0, 1, 0);
glVertex3f(0.5, -0.5, -5);
glColor3f(0, 0, 1);
glVertex3f(0,-0.75, -5);
glEnd();
glFlush();
glutSwapBuffers(); // Pentru a evita palpairea (obiectului intermitent)
// Modul de afișare cu ajutorul GLUT_DOUBLE
}
void init( void )
{
glClearColor( 1.0, 0.0, 0.0, 1.0 ); // Sterge culoarea
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(45, (GLdouble)500.0/(GLdouble)500.0, 0, 100);
glMatrixMode(GL_MODELVIEW);
}
void resize( int w, int h )
{
glViewport( 0, 0, (GLsizei) w, (GLsizei) h );
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
gluPerspective(45, (GLdouble)w/(GLdouble)h, 0, 100);
glMatrixMode( GL_MODELVIEW );
}
void myTimeOut(int id)
{
// Apelat pentru temporizator
// ...avansarea incrementala a animatiei...
rot+=5; //viteza de rotatie
glutPostRedisplay(); // solicitarea de reafisare
glutTimerFunc(100, myTimeOut, 0); // solicitarea urmatorului eveniment temporizator
}
void myKeyboard(unsigned char key,int x, int y)
{
// tasta de previzualizare marita (zoom in)
if((key=='<')||(key==',')) z_pos-=0.1f;
// tasta de previzualizare micsorata (zoom in)
if((key=='>')||(key=='.')) z_pos+=0.1f;
}
int main(int argc, char** argv)
{
glutInit(&argc,argv);
//glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("Perspectiva");
// callbacks
glutDisplayFunc(mydisplay);
glutKeyboardFunc(myKeyboard);
glutTimerFunc(100, myTimeOut, 0);
glutReshapeFunc(resize);
init();
glutMainLoop();
}
```



7.Rulaţi programul de mai jos pentru transformarea de modelare si de vizualizare.

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

GLfloat xRotated, yRotated, zRotated;
GLdouble size=1;


void display(void)
{

    glMatrixMode(GL_MODELVIEW);
    // goleste zona tampon de desenare.
    glClear(GL_COLOR_BUFFER_BIT);
    // sterge matricea de identitate.
    glLoadIdentity();
    // translateaza desenul cu z = -4.0
    // Retineti ca atunci cand z este -8.0 desenul va parea a fi mai departe sau mai mic.
    glTranslatef(0.0,0.0,-4.5);
    // Este utilizata culoarea rosie pentru a desena.
    glColor3f(0.8, 0.2, 0.1);
    // schimabrea in matricea de transformare.
    // rotatie dupa axa X
    glRotatef(xRotated,1.0,0.0,0.0);
    // rotatie dupa axa Y
    glRotatef(yRotated,0.0,1.0,0.0);
    // rotatie dupa axa X
    glRotatef(zRotated,0.0,0.0,1.0);
    // transformarea de scalare
    glScalef(1.0,1.0,1.0);
    // functie inclusa in bibloteca (glut library), pentru desenarea unui ceainic.
    glutSolidTeapot(size);
    // Golirea tamponului de afisare

    glFlush();
    // este apelat dublu buffer
   // glutSwapBuffers();
}

void reshapeFunc(int x, int y)
{
```

```
    if (y == 0 || x == 0) return;  //Nimic nu este vizibil, reintoarcere
    //Setarea unei noi matrici de proiectie
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    //Vizualizare sub unghi de:40 grade
    //Distanta de plan de aproape in perspectiva: 0.5
    //Distanta de plan de departare in perspectiva: 20.0

    gluPerspective(40.0,(GLdouble)x/(GLdouble)y,0.5,20.0);

    glViewport(0,0,x,y);  //Foloseste fereastra de randare
}

void idleFunc(void)
{

     yRotated += 0.01;

    display();
}


int main (int argc, char **argv)
{
    //Initializare GLUT
    glutInit(&argc, argv);
    //double buffering utilizat pentru a evita problemele de palpaire in animatie
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    // Dimensionare fereastra
    glutInitWindowSize(400,350);
    // Crearea ferestrei
    glutCreateWindow("Rotatia Animata a unui ceainic");
    glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
    xRotated = yRotated = zRotated = 30.0;
     xRotated=33;
     yRotated=40;
    glClearColor(0.0,0.0,0.0,0.0);
    //Atribuire functie folosita in evenimente
    glutDisplayFunc(display);
   glutReshapeFunc(reshapeFunc);
    glutIdleFunc(idleFunc);
    //Intrare in bucla de evenimente
    glutMainLoop();
    return 0;
}
```