Laborator 12

Afisarea textelor și liste de afişare

Pentru afişarea unui caracter se apeleaza functia **glutStrokeCharacter :**

```
void glutStrokeCharacter(void *font, int character);
```

- *font* specifică fontul vectorial folosit, care poate avea una din următoarele valori:
  - GLUT_STROKE_ROMAN
  - GLUT_STROKE_MONO_ROMAN
- *character* specifică caracterul ce va fi afişat

Exemplu
Funcţia **output** prezentata mai jos realizeaza afişarea unui text cu format, incepand dintr-o pozitie specificata a ferestrei curente.

```
void output(GLfloat x, GLfloat y, char *format,...)
{
  va_list args;
  char buffer[200], *p;

  va_start(args, format);
  vsprintf(buffer, format, args);
  va_end(args);
  glPushMatrix();
  glTranslatef(x, y, 0);
  for (p = buffer; *p; p++)
    glutStrokeCharacter(GLUT_STROKE_ROMAN, *p);
  glPopMatrix();
}
```

Liste de afişare

Utilizarea listelor de afişare

O listă de afişare reprezintă un grup de comenzi OpenGL stocate pentru o execuţie ulterioară. La invocarea unei liste de afişare comenzile din ea sunt executate în ordinea în care sunt întâlnite. Majoritatea comenzilor OpenGL pot fi stocate într-o listă de

afişare sau pot apare în modul imediat (sunt executate imediat). Modul imediat de programare poate fi combinat cu listele de afişare.

Listele de afişare pot îmbunătăţi programul deoarece instrucţiunile sunt stocate pentru execuţii ulterioare. Este indicat să se folosească liste de afişare în cazul în care se redesenează de mai multe ori aceeaşi figură geometrică sau dacă trebuie aplicat de mai multe ori un set de modificări de stare.

O listă de afişare este un mod eficient şi convenabil de a combina un set de comenzi OpenGL.

Exemplu: desenarea unui cerc format din 100 de segmente. Codul corespunzător desenării cercului fără a folosi liste de afişare este următorul:

```
drawCircle()
{
   GLint i;
   GLfloat cosine, sine;

   glBegin(GL_POLYGON);
      for(i=0;i<100;i++){
         cosine=cos(i*2*PI/100.0);
         sine=sin(i*2*PI/100.0);
         glVertex2f(cosine,sine);
      }
   glEnd();
}
```

Această metodă este ineficientă deoarece calculele trigonometrice vor fi efectuate de fiecare dată când cercul va fi afişat. O altă modalitate este de a salva aceste coordonate într-un vector şi a le folosi ori de câte ori este nevoie :

```
drawCircle()
{  GLint i;
   GLfloat cosine, sine;
   static GLfloat circoords[100][2];
   static GLint inited=0;

   if(inited==0){
      inited=1;
      for(i=0;i<100;i++){
         circcoords[i][0]=cos(i*2*PI/100.0);
```

```
         circcoords[i][1]=sin(i*2*PI/100.0);
      }
   }
   glBegin(GL_POLYGON);
      for(i=0;i<100;i++)
         glVertex2fv(&circcoords[i][0]);
   glEnd();
}
```

Şi această metodă prezintă dezavantajul incrementării şi testării variabilei i. Ceea ce se doreşte este să se deseneze o singură dată cercul şi să se cunoască modul de redesenare ulterior. Acest lucru este realizat prin folosirea listelor de afişare.

Exemplu: crearea unei liste de afişare

```
#define MY_CIRCLE_LIST 1

buildCircle()
{
   GLint i;
   GLfloat cosine, sine;

   glNewList(MY_CIRCLE_LIST, GL_COMPILE);
      glBegin(GL_POLYGON);
         for(i=0;i<100;i++){
            cosine=cos(i*2*PI/100.0);
            sine=sin(i*2*PI/100.0);
            glVertex2f(cosine,sine);
         }
      glEnd();
   glEndList();
}
```

Codul pentru desenarea cercului se află între apelurile **glNewList** şi **glEndList**. Aceste apeluri delimitează o listă de afişare. Argumentul **MY_CIRCLE_LIST** al funcţiei **glNewList** este un index întreg care identifică în mod unic lista de afişare. Ulterior lista de afişare poate fi executată folosind comanda **glCallList**:

```
glCallList(MY_CIRCLE_LIST);
```

Listele de afişare sunt un mod convenabil şi eficient de a vedea sub forma unui nume o secvenţă de comenzi OpenGL. O listă de afişare conţine numai apeluri OpenGL. Alte apeluri - ca în exemplul de mai sus, cum ar fi funcţiile C **cos** şi **sin** – nu sunt stocate în listele de afişare. Coordonatele şi celelalte variabile (cum ar fi conţinutul vectorului) sunt evaluate şi copiate în lista de afişare având valorile de la momentul compilării listei. După compilarea listei aceste valori nu pot fi modificate. Lista de afişare poate fi ştearsă şi se poate crea una nouă, dar o listă de afişare existentă nu poate fi editată.

Exemplu

Programul din fişierul exemplul8.c vizualizează unui tor din diferite unghiuri. Cel mai eficient mod de a face acest lucru este de a păstra torul într-o listă de afişare. Apoi, de câte ori se doreşte să se modifice poziţia observatorului se va modifica matricea ModelView şi se va executa lista de afişare pentru desenarea torului.

```
/* fişierul exemplul8.c */

#include "glut.h"

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

#define M_PI 3.14

GLuint theTorus;

/* afişare tor */
void torus(int numc, int numt)
{
   int i, j, k;
   double s, t, x, y, z, twopi;

   twopi = 2 * (double)M_PI;
   for (i = 0; i < numc; i++) {
      glBegin(GL_QUAD_STRIP);
      for (j = 0; j <= numt; j++) {
         for (k = 1; k >= 0; k--) {
            s = (i + k) % numc + 0.5;
            t = j % numt;
```

```
            x = (1+.1*cos(s*twopi/numc))*cos(t*twopi/numt);
            y = (1+.1*cos(s*twopi/numc))*sin(t*twopi/numt);
            z = .1 * sin(s * twopi / numc);
            glVertex3f(x, y, z);
         }
      }
      glEnd();
   }
}

/* creează lista de afişare pentru tor */
void init(void)
{
   theTorus = glGenLists (1);
   glNewList(theTorus, GL_COMPILE);
   torus(8, 25);
   glEndList();

   glShadeModel(GL_FLAT);
   glClearColor(0.0, 0.0, 0.0, 0.0);
}

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);
   glCallList(theTorus);
   glFlush();
}

void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(30, (GLfloat) w/(GLfloat) h, 1.0, 100.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);
```

```
}

/* la apăsarea tastei 'x' - se roteşte în jurul axei x
   la apăsarea tastei 'y' - se roteşte în jurul axei y
   la apăsarea tastei 'i' - se poziţioneaza torul în poziţia
originală
*/
void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
   case 'x':
   case 'X':
      glRotatef(30.,1.0,0.0,0.0);
      glutPostRedisplay();
      break;
   case 'y':
   case 'Y':
      glRotatef(30.,0.0,1.0,0.0);
      glutPostRedisplay();
      break;
   case 'i':
   case 'I':
      glLoadIdentity();
      gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);
      glutPostRedisplay();
      break;
   case 27:
      exit(0);
      break;
   }
}

int main(int argc, char **argv)
{
   glutInitWindowSize(200, 200);
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutCreateWindow(argv[0]);
   init();
   glutReshapeFunc(reshape);
```

```
   glutKeyboardFunc(keyboard);
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```

Utilizatorul poate roti torul în jurul axei OX sau OY prin apăsarea tastelor x respectiv y. De câte ori se întamplă acest lucru este apelată funcţia callback **keyboard** care înmulţeşte matricea de rotaţie de 30^o în jurul axei x sau y cu matricea curentă ModelView, după care este apelată funcţia **glutPostRedisplay,** care face ca funcţia **glutMainLoop** să apeleze funcţia **display** şi să afişeze torul după prelucrarea altor evenimente. La apăsarea tastei 'i' funcţia **keyboard** reface matricea iniţială ModelView şi reafişează torul în poziţia sa iniţială. Funcţia **display** şterge fereastra şi apelează funcţia **glCallList** pentru a executa comenzile din lista de afişare.

Dacă nu s-ar fi folosit liste de afişare funcţia **display** ar fi trebuit să conţină comenzi de desenare a torului de fiecare dată când ar fi fost apelată.

O listă de afişare conţine numai comenzi OpenGL. În exemplul din fişierul exemplul8.c sunt stocate apelurile funcţiilor **glBegin**, **glVertex** şi **glEnd**. Parametrii apelurilor sunt evaluaţi şi valorile lor sunt copiate în lista de afişare la crearea sa. Toate calculele trigonometrice pentru crearea torului sunt făcute o singură dată ceea ce duce la creşterea performanţelor afişării.

Exemplu: aplicarea unor transformări unor obiecte geometrice şi apoi desenarea rezultatului:

```
glNewList(1, GL_COMPILE);
afiseaza_obiectele_geometrice();
glEndList();

glLoadMatrix(M);
glCallList(1);
```

Dacă obiectele sunt transformate de fiecare dată în acelaşi mod este bine să se păstreze matricea de transformare într-o listă de afişare.

Exemplu

In unele implementări, se vor putea îmbunătaţi performanţele prin transformarea obiectelor în momentul definirii lor în loc de a le transforma de fiecare dată când sunt afişate :

```
glNewList(1, GL_COMPILE);
glLoadMatrix(M);
afiseaza_obiectele_geometrice();
glEndList();

glCallList(1);
```

Listele de afişare au şi dezavantaje. Listele foarte mici nu vor îmbunătăţi execuţia programului datorită overhead-ului execuţiei listei. Un alt dezavantaj constă din faptul că o listă de afişare nu poate fi modificată şi conţinutul său nu poate fi citit. Dacă aplicaţia necesită păstrarea datelor separat faţă de lista de afisare atunci va fi necesară memorie suplimentară.

. Crearea şi executarea listelor de afişare

Funcţiile **glNewList** şi **glEndList** sunt folosite pentru delimitarea unei liste de afişare, care este executată prin apelul funcţiei **glCallList** având ca parametru identificatorul său. În fişierul exemplul9.c lista de afişare este creată în funcţia **init**. În funcţia **display** lista de afişare va fi apelată de 10 ori. Listele de afişare alocă memorie pentru a stoca comenzile şi valorile oricărei variabilele necesare. Funcţia **glTranslatef** din lista de afişare modifică poziţia următorului obiect ce va fi afişat. Apelul **drawLine** este de asemenea afectat de funcţia **glTranslatef** care o precede.

```
/* fişierul exemplul9.c */

#include "glut.h"
#include <stdlib.h>

GLuint listName;

void init (void)
{
   listName = glGenLists (1);
   glNewList (listName, GL_COMPILE);
```

```
      glColor3f (1.0, 0.0, 0.0);  /*  culoarea curentă roşu  */
      glBegin (GL_TRIANGLES);
      glVertex2f (0.0, 0.0);
      glVertex2f (1.0, 0.0);
      glVertex2f (0.0, 1.0);
      glEnd ();
      glTranslatef (1.5, 0.0, 0.0); /*  modificare poziţie  */
   glEndList ();
   glShadeModel (GL_FLAT);
}
void drawLine (void)
{
   glBegin (GL_LINES);
   glVertex2f (0.0, 0.5);
   glVertex2f (15.0, 0.5);
   glEnd ();
}

void display(void)
{
   GLuint i;

   glClear (GL_COLOR_BUFFER_BIT);
   glColor3f (0.0, 1.0, 0.0);  /*  culoarea curentă verde  */
   for (i = 0; i < 10; i++)    /*  afişare 10 triunghiuri  */
      glCallList (listName);
   drawLine ();
   glFlush ();
}

void reshape(int w, int h)
{
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w <= h)
      gluOrtho2D (0.0, 2.0, -0.5 * (GLfloat) h/(GLfloat) w,
         1.5 * (GLfloat) h/(GLfloat) w);
   else
      gluOrtho2D (0.0, 2.0*(GLfloat) w/(GLfloat) h, -0.5, 1.5);
```

```
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

void keyboard(unsigned char key, int x, int y)
{
   switch (key)
   {
      case 27:
         exit(0);
   }
}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(650, 50);
   glutCreateWindow(argv[0]);
   init ();
   glutReshapeFunc (reshape);
   glutKeyboardFunc (keyboard);
   glutDisplayFunc (display);
   glutMainLoop();
   return 0;
}
```

Observaţie: Dacă o listă de afişare conţine comenzi de transformare trebuie avut grijă care va fi efectul acestora în program mai târziu.

La un moment dat poate fi creată numai o singură lista de afişare. Cu alte cuvinte trebuie terminată crearea unei liste de afişare (**glNewList şi glEndList**) înainte de a crea o altă listă. Apelul funcţiei **glEndList** fără a fi apelat înainte funcţia **glNewList** va genera eroarea **GL_INVALID_OPERATION**.

Crearea unei liste de afişare

Fiecare listă de afişare este identificată printr-un index întreg. La crearea unei liste de afişare trebuie să se aibă grijă să nu se aleagă un index care este deja folosit, altfel se va şterge lista de afişare existentă. Pentru evitarea acestui lucru se va folosi funcţia **glGenLists** ce va genera unul sau mai mulţi indici nefolosiţi :

```
GLuint glGenLists(GLsizei range);
```

Funcţia alocă un domeniu de *range* numere continue nefolosite ca indici pentru liste de afişare. Valoarea întoarsă de funcţie reprezintă indicele de început a blocului de indici nefolosiţi. Indicii returnaţi vor fi marcaţi ca folosiţi astfel ca apelurile ulterioare ale funcţiei **glGenLists** nu vor întoarce aceşti indici până nu vor fi şterşi. Funcţia întoarce 0 dacă nu este disponibil numarul de indici ceruţi sau dacă *range* este 0.

Exemplu: alocarea unui singur index; dacă el este liber va fi folosit pentru a crea o nouă listă de afişare :

```
listIndex = glGenLists(1);
if (listIndex != 0) {
   glNewList(listIndex,GL_COMPILE);
      ...
   glEndList();
}
```

Observaţie: Indexul 0 nu reprezintă un index valid pentru lista de afişare.

```
void glNewList (GLuint list, GLenum mode);
```

Funcţia specifică începerea unei liste de afişare. Funcţiile OpenGL care vor fi apelate (până la întâlnirea funcţiei **glEndList** care marchează sfârşitul listei de afişare) sunt stocate în lista de afişare, excepţie facând câteva funcţii OpenGL care nu pot fi stocate. Aceste funcţii restricţionate sunt executate imediat în timpul creării listei de afişare.

- *list* este un întreg pozitiv diferit de 0 care identifică în mod unic lista de afişare.
- valorile posibile ale parametrului *mode* sunt **GL_COMPILE** şi **GL_COMPILE_AND_EXECUTE**. Se foloseşte valoarea **GL_COMPILE** dacă nu se doreşte executarea comenzilor OpenGL la plasarea lor în lista de afişare; pentru execuţia imediată în momentul plasării în lista de afişare, pentru folosiri ulterioare, se va secifica parametrul **GL_COMPILE_AND_EXECUTE**.

```
void glEndList (void);
```

Funcţia marchează sfârşitul unei liste de afişare.

La crearea unei liste de afişare ea va fi stocată împreună cu contextul OenGL curent. Astfel, dacă contextul va fi distrus, de asemenea lista de afişare va fi distrusă. În unele sisteme este posibil ca listele de afişare să partajeze contexte multiple. În acest caz lista de afişare va fi distrusă în momentul în care ultimul context din grup este distrus.

La crearea unei liste de afişare în ea vor fi stocate doar valorile expresiilor. Dacă valorile dintr-un vector sunt modificate, valorile din lista de afişare nu vor fi modificate.

Exemplu

Se crează o listă de afişare ce conţine o comandă care setează culoarea curentă de afişare negru (0.0, 0.0, 0.0). Modificarea ulterioară a valorii vectorului **color_vector** în roşu (1.0, 0.0, 0.0) nu are nici un efect asupra listei de afişare deoarece aceasta conţine valorile de la crearea sa.

```
GLfloat color_vector[3] = {0.0, 0.0, 0.0};
glNewList(1, GL_COMPILE);
   glColor3fv(color_vector);
glEndList();
color_vector[0] = 1.0;
```

Într-o listă de afişare nu pot fi stocate şi executate orice comenzi OpenGL. În tabelul II.8 sunt prezentate comenzile care nu pot fi stocate într-o listă de afişare (apelul funcţiei **glNewList** în timpul creării unei liste de afişare generează eroare).

glColorPointer()	glFlush()	GlNormalPointer()
glDeleteLists()	glGenLists()	GlPixelStore()
glDisableClientState()	glGet*()	GlReadPixels()
glEdgeFlagPointer()	glIndexPointer()	GlRenderMode()
glEnableClientState()	glInterleavedArrays()	GlSelectBuffer()
glFeedbackBuffer()	glIsEnabled()	GlTexCoordPointer()
glFinish()	glIsList()	GlVertexPointer()

Tabelul II.8.

La folosirea unui sistem OpenGL în reţea, clientul poate rula pe o maşină şi server-ul pe alta. După crearea unei liste de afişare ea se va afla la server astfel că server-ul nu se poate baza pe client pentru nici o informaţie relativă la lista de afişare. Astfel, orice comandă care întoarce o valoare nu poate fi stocată într-o listă de afişare. În plus, comenzile care modifică starea clientului cum ar fi **glPixelStore**, **glSelectBuffer** şi comenzile de definire a vectorilor de vârfuri nu pot fi stocate într-o listă de afişare.

Operaţiile unor comenzi OpenGL depind de starea clientului. De exemplu, funcţiile de specificare a vârfurilor vectorilor (cum ar fi **glVertexPointer, glColorPointer** şi **glInterleavedArrays**) setează pointeri de stare ai clientului şi nu pot fi stocate într-o listă de afişare. Funcţiile **glArrayElement**, **glDrawArrays** şi **glDrawElements** transmit date către server pentru a construi primitive din elementele vectorilor. Astfel de operaţii pot fi stocate într-o listă de afişare.

Vectorul de vârfuri stocat în lista de afişare este obţinut prin dereferenţierea datei din pointeri nu prin stocarea pointerilor . Astfel, modificările datelor din vectorul de vârfuri nu va afecta definirea primitivei în lista de afişare.

Funcţii cum ar fi **glFlush** şi **glFinish** nu pot fi stocate într-o listă de afişare deoarece depind de starea clientului în momentul execuţiei.

Execuţia unei liste de afişare

După crearea unei liste de afişare aceasta poate fi executată prin apelul funcţiei **glCallList**. O listă de afişare poate fi executată de mai multe ori şi de asemenea o listă de afişare poate fi executată îmbinându-se cu apeluri în modul imediat.

```
void glCallList (GLuint list);
```

Functia execută lista de afişare specificată de parametrul *list*. Comenzile din lista de afişare sunt executate în ordinea în care au fost salvate ca şi cum ar fi executate fără a se folosi o lista de afişare. Dacă lista nu a fost definită nu se va întâmpla nimic.

Funcţia **glCallList** poate fi apelată din orice punct al programului atâta timp cât contextul OpenGL care accesează lista de afişare este activ (contextul care a fost activ la crearea listei de afişare sau un context din acelaşi grup partajat). O listă de afişare poate fi creată într-o funcţie şi executată în altă funcţie atâta timp cât indexul său o identifică în mod unic. De asemenea, contextul unei liste de afişăre nu poate fi salvat într-un fişier şi nici nu se poate crea o listă de afişare dintr-un fişier. În acest sens o listă de afişare este proiectată pentru a fi folosită temporar.

Liste de afişare ierarhice

O listă de afişare ierarhică este o listă de afişare care execută o altă listă de afişare prin apelul funcţiei **glCallList,** între perechile de funcţii **glNewList** şi **glEndList**. O listă de afişare ierarhică este folositoare pentru un obiect alcătuit din componente, în mod special dacă aceste componente sunt folosite de mai multe ori.

Exemplu: o listă de afişare care desenează o bicicletă prin apelul altor liste de afişare pentru desenarea componentelor :

```
glNewList(listIndex,GL_COMPILE);
   glCallList(handlebars);
   glCallList(frame);
   glTranslatef(1.0,0.0,0.0);
   glCallList(wheel);
   glTranslatef(3.0,0.0,0.0);
   glCallList(wheel);
glEndList();
```

Pentru evitarea recursivităţii infinite limita nivelului de imbricare al listelor de afişare este 64, dar această limită depinde de implementare. Pentru a determina limita specifică implementării OpenGL pe care se lucrează se apelează funcţia :

```
glGetIntegerv(GL_MAX_LIST_NESTING, GLint *data);
```

OpenGL permite crearea unei liste de afişare care să apeleze o altă listă de afişare care nu a fost încă creată. Nu va avea nici un efect dacă prima listă o apelează pe cea de a doua care încă nu a fost definită.

Exemplu : listă de afişare ierarhică

```
glNewList(1,GL_COMPILE);
   glVertex3f(v1);
glEndList();
glNewList(2,GL_COMPILE);
   glVertex3f(v2);
glEndList();
glNewList(3,GL_COMPILE);
   glVertex3f(v3);
```

```
glEndList();

glNewList(4,GL_COMPILE);
   glBegin(GL_POLYGON);
      glCallList(1);
      glCallList(2);
      glCallList(3);
   glEnd();
glEndList();
```

Pentru afişarea poligonului se apelează lista de afişare 4. Pentru a edita un vârf este necesar să se creeze din nou lista de afişare corespunzătoare vârfului respectiv. Deoarece un index identifică în mod unic lista de afişare, crearea unei alte liste cu acelaşi index o va şterge în mod automat pe cea existentă. Trebuie reţinut că această metodă nu foloseşte în mod optim memoria şi nu îmbunătăţeşte performanţele, dar este folositoare în unele cazuri.

Gestiunea listelor de afişare cu indici

Pentru a obţine un indice nefolosit care să identifice o nouă listă de afişare se poate folosi funcţia **glGenLists**. Dacă nu se doreşte folosirea acestei funcţii atunci se poate folosi **glIsList** pentru a determina dacă un anumit index este folosit.

```
GLboolean glIsList(GLuint list);
```

Funcţia întoarce **GL_TRUE** dacă indexul specificat de parametrul *list* este deja folosit pentru o listă de afişare şi **GL_FALSE** în caz contrar.

Pentru a şterge în mod explicit o listă de afişare sau un domeniu continuu de liste se foloseşte funcţia **glDeleteLists**. Folosirea funcţiei **glDeleteLists** eliberează indicii corespunzători listelor şterse să fie disponibili din nou.

```
void glDeleteLists(GLuint list, GLsizei range);
```

Funcţia şterge *range* liste de afişare începând de la indexul specificat de *list.* Este ignorată încercarea de a se şterge o listă de afişare care nu a fost creată.

Execuţia listelor de afişare multiple

OpenGL furnizează un mecanism eficient de a executa succesiv liste de afişare. Acest mecanism necesită introducerea indicilor listelor de afişare într-un vector si apoi apelarea funcţiei **glCallLists**. O utilizare pentru acest mecanism este acela de a crea un font şi fiecare indice de listă de afişare să corespundă valorii ASCII a caracterului din font. Pentru a avea mai multe fonturi este necesară stabilirea unui index iniţial diferit pentru fiecare font. Acest index iniţial poate fi specificat prin apelarea funcţiei **glListBase** înainte de a apela **glCallLists**.

```
void glListBase(GLuint base);
```

Funcţia specifică offset-ul care este adunat indicilor listelor de afişare în apelul funcţiei **glCallLists** pentru a obţine indicii listelor de afişare finali. Valoarea implicită a parametrului *base* este 0. Parametrul *base* nu are nici un efect asupra apelului **glCallList,** care execută o singură listă de afişare, sau asupra funcţiei **glNewList**.

```
void glCallLists(GLsizei n, GLenum type, const GLvoid *lists);
```

Functia execută *n* liste de afişare. Indicii listelor care vor fi executate vor fi calculaţi prin adunarea offset-ului indicat de baza curentă a listei de afişare (specificat cu ajutorul funcţiei **glListBase** la valorile întregi indicate de parametrul *lists.*
Parametrul *type* specifică tipul valorilor din *lists.* El poate avea una din valorile **GL_BYTE**, **GL_UNSIGNED_BYTE**, **GL_SHORT**, **GL_UNSIGNED_SHORT**, **GL_INT**, **GL_UNSIGNED_INT** sau **GL_FLOAT**, adică ceea ce indică *lists* poate fi tratată ca un vector de bytes, unsigned bytes, shorts, unsigned shorts, integers, unsigned integers, sau floats. De asemenea parametrul *type* poate avea una din valorile **GL_2_BYTES**, **GL_3_BYTES** sau **GL_4_BYTES** caz în care succesiuni de 2, 3 sau 4 octeţi vor fi citiţi din *lists* şi apoi sunt deplasaţi şi adunaţi octet cu octet pentru a calcula offset-ul listei de afişare. Pentru aceasta este folosit următorul algoritm (byte[0] reprezintă începutul secvenţei de octeţi).

```
/* b = 2, 3 sau 4; octeţii sunt numerotaţi în vector 0,1,2,3 */
 offset = 0;
 for (i = 0; i < b; i++) {
    offset = offset << 8;
    offset += byte[i];
```

```
 }
 index = offset + listbase;
```

Exemplu
Definirea listelor de afişare multiple: afişarea caracterelor dintr-un set de caractere vectorial

```
void initStrokedFont(void)
/* setează indicii listelor de afişare pentru fiecare caracter
   corespunzător valorii lor ASCII */
{
   GLuint base;

   base = glGenLists(128);
   glListBase(base);
   glNewList(base+'A', GL_COMPILE);
      drawLetter(Adata); glEndList();
   glNewList(base+'E', GL_COMPILE);
      drawLetter(Edata); glEndList();
   glNewList(base+'P', GL_COMPILE);
      drawLetter(Pdata); glEndList();
   glNewList(base+'R', GL_COMPILE);
      drawLetter(Rdata); glEndList();
   glNewList(base+'S', GL_COMPILE);
      drawLetter(Sdata); glEndList();
   glNewList(base+' ', GL_COMPILE);    /* spaţiu */
      glTranslatef(8.0, 0.0, 0.0);
   glEndList();
}
```

Funcţia **glGenLists** alocă 128 de indici continui pentru listele de afişare. Primul indice alocat devine baza listei de afişare. Pentru fiecare caracter va fi creată câte o listă de afişare; fiecare index al listei de afişare este suma dintre indicele de bază şi valoarea ASCII a literei. În acest exemplu sunt create numai câteva litere şi caracterul ‘ ‘. După crearea listelor de afişare poate fi apelata funcţia **glCallLists** pentru a le executa.

Exemplu
Apelul functiei **printStrokedString** având ca parametru un caracter :

```
void printStrokedString(GLbyte *s)
{
   GLint len = strlen(s);
   glCallLists(len, GL_BYTE, s);
}
```

Gestiunea variabilelor de stare folosind liste de afişare

O listă de afişare poate conţine apeluri care să modifice valorile variabilelor de stare OpenGL. Aceste valori se modifică la execuţia listei de afişare (ca şi în modul imediat de execuţie al comenzilor) şi modificările se păstrează şi după execuţia completă a listei de afişare.

Exemplu
Modificările culorii curente şi a matricii curente făcute în timpul execuţiei listei de afişare rămân vizibile şi după executarea sa:

```
glNewList(listIndex,GL_COMPILE);
   glColor3f(1.0, 0.0, 0.0);
   glBegin(GL_POLYGON);
      glVertex2f(0.0,0.0);
      glVertex2f(1.0,0.0);
      glVertex2f(0.0,1.0);
   glEnd();
   glTranslatef(1.5,0.0,0.0);
glEndList();
```

Dacă se va apela următoarea secvenţă de cod, segmentul de dreaptă desenat după lista de afişare va avea culoarea roşie (culoarea curentă) şi va fi translatat cu (1.5, 0.0, 0.0):

```
glCallList(listIndex);
glBegin(GL_LINES);
   glVertex2f(2.0,-1.0);
   glVertex2f(1.0,0.0);
glEnd();
```

Uneori este necesar ca modificările stării să fie păstrate, dar alteori după execuţia unei liste de afişare se doreşte să se revină la starea anterioară. Într-o listă de afişare nu

poate fi folosită funcţia **glGet***, aşa că trebuie folosit un alt mod de a interoga şi stoca valorile variabilelor de stare. În acest scop poate fi folosită funcţia **glPushAttrib** pentru a salva un grup de variabile de stare şi **glPopAttrib** pentru a reface variabilele.

Exemplu
Refacerea variabilelor de stare din interiorul unei liste de afişare

```
glNewList(listIndex,GL_COMPILE);
   glPushMatrix();
   glPushAttrib(GL_CURRENT_BIT);
   glColor3f(1.0, 0.0, 0.0);
   glBegin(GL_POLYGON);
      glVertex2f(0.0,0.0);
      glVertex2f(1.0,0.0);
      glVertex2f(0.0,1.0);
   glEnd();
   glTranslatef(1.5,0.0,0.0);
   glPopAttrib();
   glPopMatrix();
glEndList();
```

Exemplu
Dacă se foloseşte lista de afişare de mai sus atunci se va desena un segment de dreaptă de culoare verde şi netranslatat:

```
void display(void)
{
   GLint i;
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(0.0, 1.0, 0.0);  /* setează culoarea curentă verde
*/
   for (i = 0; i < 10; i++)
      glCallList(listIndex);  /* lista de afişare apelată de 10
ori*/
   drawLine();        /* unde şi cum apare această linie? */
   glFlush();
}
```

Încapsularea schimbărilor de mod

Listele de afişare pot fi folosite pentru a organiza şi stoca grupuri de comenzi, pentru a modifica diferite moduri sau a seta diferiţi parametri. Când se doreşte comutarea de la un grup de setări la un altul folosirea listelor de afişare poate fi mai eficientă decât apelarea directă.

Listele de afişare pot fi mai eficiente decât modul imediat pentru comutarea între diferite setări ale surselor de lumină, modelelor de iluminare şi parametrilor de material. De asemenea, listele de afişare se pot folosi şi pentru generarea liniilor cu şablon, precum şi pentru ecuaţiile planului de decupare. În general, execuţia listelor de afişare este cel puţin tot atât de rapidă ca şi apelul direct dar în cazul folosirii listelor de afişare este introdus un overhead.

Exemplu

Folosirea listelor de afişare pentru a comuta între trei şabloane diferite de linii. La început se apelează funcţia **glGenLists** pentru a aloca o listă de afişare pentru fiecare şablon. Apoi se foloseşte funcţia **glCallList** pentru a comuta între un şablon şi altul.

```
GLuint offset;
offset = glGenLists(3);
glNewList (offset, GL_COMPILE);
    glDisable (GL_LINE_STIPPLE);
glEndList ();
glNewList (offset+1, GL_COMPILE);
    glEnable (GL_LINE_STIPPLE);
    glLineStipple (1, 0x0F0F);
glEndList ();
glNewList (offset+2, GL_COMPILE);
    glEnable (GL_LINE_STIPPLE);
    glLineStipple (1, 0x1111);
glEndList ();
#define drawOneLine(x1,y1,x2,y2) glBegin(GL_LINES); \
    glVertex2f ((x1),(y1)); glVertex2f ((x2),(y2)); glEnd();
glCallList (offset);
drawOneLine (50.0, 125.0, 350.0, 125.0);
glCallList (offset+1);
drawOneLine (50.0, 100.0, 350.0, 100.0);
glCallList (offset+2);
drawOneLine (50.0, 75.0, 350.0, 75.0);
```