eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Konputagailu Bidezko Grafikoak: objektuei **aldaketak** eragin

Informatikako Ingenieritza

Konputazioa

Abstract

Fase honetan objektuen aldaketak inplementatu behar dira; dokumentu honek inplementazioari buruzko zenbait azalpen jasotzen du. Fase honetan, gutxienez, hiru aldaketa inplementatu behar dira, hala nola, translazioak, biraketak eta tamaina aldaketak. Objektu zerrenda bat daukagu gure mundu birtualean, eta aldaketak objektu horien artean aukeratuta daukagunari eragingo zaizkio une bakoitzean. Edozein momentutan objektuz alda dezake erabiltzaileak, beraz, une horretatik aurrerako aldaketak objektu berriari eragingo zaizkio. Bestalde, aldaketak desegiteko aukera ere eskaini behar du aplikazioak. Desegiteak aukeratutako objektuari aurretik eragin zaizkion aldaketak desegingo ditu. Teorian ikusi duzuen legez, aldaketak egiteko objektuaren koordenatuak 4×4 tamainako matrize batekin biderkatu behar ditugu.

1 OpenGLren matrizeak

OpenGLk zenbait matrize erabiltzen ditu. Horietako bat GL_MODELVIEW deritzona da. Objektu bat sisteman sartzean, bere koordenatuak matrize horrekin biderkatzen dira. Hortaz, objektuen aldaketak kudeatzeko matrize hori erabiliko dugu. Aurreko fasean objektuen marrazketari ekin genion, ondoko kodea erabiliz:

```
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
/* First , we draw the axes */
draw_axes();
/* Now each of the objects in the list */
while ( aux_obj != 0)
  {
  /* Select the color, depending on whether the current
     object is the selected one or not */
  ...
  /* Draw the object;
     for each face create a new polygon with the corresponding vertices */
  glLoadIdentity();
  for (f = 0; f < aux_obj->num_faces; f++)
    {
    glBegin(GL_POLYGON);
    for (v = 0; v < aux_obj->face_table[f].num_vertices; v++)
      {
      v_index = aux_obj->face_table[f].vertex_table[v];
      glVertex3d (aux_obj->vertex_table[v_index].coord.x,
                  aux_obj->vertex_table[v_index].coord.y ,
                  aux_obj->vertex_table[v_index].coord.z );
      }
    glEnd();
    }
  aux_obj = aux_obj->next ;
  }
```

Goiko kodean, lehenengo bi lerroetan **glMatrixMode** eta **glLoadIdentity** funtzioak ikusten ditugu. Lehenengoak OpenGLren zein matrize (GL MODELVIEW kasu honetan) erabiliko den esateko balio du. Une horretatik aurrera, matrizeekin egingo diren eragiketak GL_MODELVIEW matrizeari eragingo diote. Ondorioz, bigarren aginduak aukeratutako matrizean identitatea kargatuko du.

Bi agindu horien ondorioz, erpin baten koordenatuak glVertex3d aginduaren bidez sartzerakoan, OpenGL-k identitate matrizea biderkatuko die koordenatu horiei, eta dauden horretan lagako ditu.

while begiztaren barruan, objektu bakoitza marraztu aurretik (bere poligonoak marrazteko agingu aurretik, alegia) berriro ere identitate matrizea kargatzen dugu, matrizean legokeen edozein aldaketa ezerezean uzteko.

Marraztu behar dugun objektuari aldaketarik eragin nahi izanez gero, matrizea hasieratu ondoren guk nahi dugun matrizea sartu beharko genuke, **glMultMatrixd** funtzioa erabiliz, edota **glLoadMatrixd** funtzioa erabiliz; funtzio hauek parametro bakarra dute, **GLdouble *** bat, eta erakusle horrek erabili nahi dugun matrizearen hasierara apuntatu behar du (matrizearen hasiera erakutsi behar du).

OpenGLn erabiltzen diren matrizeak 16 GLdoublez osatutako bektoreak dira. Matrize berri bat sortzeko ondoko kodea erabili behar da:

```
GLdouble *matrizea;

matrizea = malloc(sizeof(GLdouble)*16);
```

Bektorearen 16 elementu horietan erabili nahi dugun matrizearen balioak sartu behar dira. Hori ondo egiteko, matrizeak zutabeka definitzen direla jakin behar da (*column major*). Hau da, matrizearen elementu bakoitza bektore horretan honako ordena honetan kokatuta dago:

$$\begin{pmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{pmatrix}$$

Eta sortu nahi dugun matrizea hurrengo hau balitz:

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix}$$

jarraian dagoen kodea idatzi beharko genuke:

```
m[0]= a; m[4]= b; m[8] = c; m[12]= d;
m[1]= e; m[5]= f; m[9] = g; m[13]= h;
m[2]= i; m[6]= j; m[10]= k; m[14]= l;
m[3]= m; m[7]= n; m[11]= o; m[15]= p;
```

Matrize hori erabili nahiko bagenu objektua aldatzeko, OpenGL-ren MODELVIEW matrizean kopiatu beharko genuke, eta horretarako bi aukera dauzkagu:

1. zuzenean korgatu dezakegu

   ```
   glMatrixMode(GL_MODELVIEW);
   glLoadMatrixd(matrizea);
   ```

2. Identitatea kargatu eta hari matrizea biderkatu diezaiokegu, alegia,

   ```
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glMultMatrixd(matrizea);
   ```

2 Objektu bat, aldaketa pila bat

Kontutan hartu behar da objektu bakoitzak aldaketa asko jasan ditzakeela, eta gainera, aldaketa horiek desegiteko aukera izan behar dugu.

Hori dena inplementatzeko erarik errazena objektu bakoitzari aldaketa zerrenda bat esleitzea da; zerrenda hori datu egitura berria izango da, eta zerrenda horretan objektuak izan dituen aldaketa-matrizeak jasoko ditugu. Beraz, objektua3d egiturari eremu berria gehitu behar zaio, eta eremu hori

hasieratu egin behar da. Hasieran, hau da, objektua fitxategitik kargatu bezain pronto, objektuari matrize bakarra daukan zerrenda erantsiko diogu, identitate matrizea besterik ez daukan zerrenda. Alegia, elementu batentzat memoria erreserbatu beharko da eta bere ondoren matrize gehiago ez daudela ere jaso beharko dugu.

Matrize zerrendaren helburua da lehenengo matrizea erabiltzea objektua marrazterakoan, hau da, zerrendako burua erabiliko dugu objektua aldatzeko. Objektua kargatzerakoan identitatea jarri dugunez, objektua bere horretan marraztuko da hasieran.

Objektuari aldaketa bat eragiten diogun bakoitzean, objektuaren matrizeari aldaketa adierazten duen matrizea biderkatu behar zaio, eta biderketaren emaitza izango da objektua aldatzeko erabiliko dugun matrize berria, beraz, objektuari lotutako matrize zerrendan elementu berri bat sartu beharko dugu, eta elementu berri horren hurrengoa izango da aldaketaren aurretik objektuak zeukan lehenengo elementua.

Objektua kargatu bezain pronto identitatea sartuko dugu zerrendan, ondoren, aldaketak eragin ahala $M_1, M_2, M_3 \dots M_k$ matrizeak sartuko ditugu; matrize horietako bakoitza lortuko dugu aurreko egoerako matrizeari M_a aldaketa-matrizea biderkatuz, hau da, $M_i = M_{i-1} * M_a$ edo $M_i = M_a * M_{i-1}$ izango da (kontuan izan matrizeen arteko biderketan ordenak eragina duela).

Zerrenda buruan beti azken matrizea jasoko dugu, eta matrize hori erabiliko dugu objektua marrazteko. Horrela, azken aldaketa desegiteko nahikoa da zerrendaren buruan dagoen matrizea kentzea. bigarren zegoen matrizea jarriko da buruan, eta matrize hori da objektuaren aurreko egoera zehazten duena.

3 Interfazearen funtzionamendua

Gure programak egoera-makina moduan funtzionatuko du, beti egoeraren batean egon behar du, eta hasierako egoera zein den dokumentazioan zehaztu behar da. Egoera makinen logika jarrai dezan, zerbait egiterakoan egoera zein den kontuan hartuko du gure aplikazioak, esate baterako, geziren bat sakatzen badu erabiltzaileak, horren eragina egoera-aldagaien araberakoa izango da: aktibo dagoen elementua objektua dalitz (hiru elementu mota aktibatu ahal izango ditu aplikazioak, objektuak, kamerak eta argiak), eta aldaketa-mota posibleen artean (biraketa, traslazioa edota eskala aldaketa dira aukerak) biraketa balego aktibo, orduan, aukeratutako objektuari geziak adierazitako biraketa eragingo dio aplikazioak.

Dena den, egoera-aldagaien balioak edozein unetan alda daitezke: esate baterako, e, r edo t sakatuz aldaketa-mota aktiboa zein den zehaztu daiteke,

eskala aldaketa, biraketa edota traslazioa izatea behartzen dute hurrenez hurren. Hortik aurrera, geziekin *AvPag* edota *RePag* teklekin eragindako aldaketak egoera aldagaiak adierazitakoa izango da.

Beste horrenbeste esan daiteke elementuari buruz, alegia, objektei (**o** tekla), kamerei (**k** tekla), edota argiei (**l** tekla), eragingo zaie aldaketa.

Azkenik, erreferentzia sistema ere egoera aldagai batek zehaztuko du: lokala ala globala.

4 Egin beharreko aldaketak

Hasierako programatxoari aldaketak eragiteko aukera gehitu nahi badiogu aldaketa asko egin behar dira:

1. Egoera aldagaiak gehitu. Gutxienez hiru egoera-aldagai behar ditugu: Alde batetik, aldaketa mota adieraziko duen aldagaia behar dugu. Aldagai honek hiru balio eduki ditzake, **biraketa**, **traslazioa** ala **neurri aldaketa** eragin behar den esango baitigu.

   Bigarren aldagaiak erreferentzia-sistema zein den adieraziko du, alegia, **munduko erreferentzia-sistema** ala **erreferentzia-sistema lokala** erabiliko dugun esango digu.

   Hirugarren aldagaiak adieraziko du zein elementu den aldatuko duguna. Oraingoz objektuak aldatuko ditugu, baina aurrera begira kamerak eta argiak ere aldatu ahal izango ditugu, beraz, hirugarren aldagai honek ere hiru balio izango ditu: **objektua**, **kamera** edo **argi-iturria** aldatu behar den honek esango baitu.

   Hiru aldagai hauek programa nagusian **definitu** eta **hasieratu** behar dira, baina edonon erabili daitezkeenez, *definiz.h* fitxategian **extern** bezala deklaratzea komeni da.

2. Objektu bakoitzari matrize zerrenda gehitu eta identitatearekin hasieratu. Horretarako matrize zerrenda definitu eta hasieratu behar da. Definitzeko *definiz.h* fitxategian matrize zerrendarentzat datu-egitura definitu eta object3d egiturari eremu bat gehitu behar zaio, eremu horrek lehenengo matrizea erakustsi behar du.

```
/***************************
 * Structure to store a     *
 * list of Matrixes         *
 ***************************/
typedef struct MZ
    {
    double M[16];
    struct MZ *hptr;     // next matrix element
```

```
    } MZ:

/***************************
 * Structure to store a    *
 * pile of 3D objects      *
 ***************************/
struct object3d{
    GLint num_vertices;      /* number of vertices in the object*/
    ...
    MZ *MZptr;               /* pointer to the head of the list of matrixes */
    }
```

Horrez gain, objektua kargatu bezain pronto erakuslea matrizea jasoko duen elementuari apuntatzen jarri behar da, hau da, elementu batentzat memoria erreserbatu eta bertan identitatea kargatu, eta bere ondoren matrize gehiago ez dagoela esan behar zaio. Hau *io.c* fitxategian, keyboard funtzioaren kodean aldatu behar da, hain zuzen ere fitxategitik irakurtzeko kodean (*case 'f':*)

3. Tekla bereziei arreta eskainiko dien funtzioa: Gezien bidez, *RePag* teklaren bidez eta *AvPag* teklaren bidez aginduko du erabiltzaileak aldaketaren bat eragin nahi duela. Tekla hauek ez dute aktibatzen teklatuari arreta ematen dion funtzioa, beste funtzio bat aktibatzen dute, eta funtzio hori zein den eta zer egiten duen adierazi behar zaio gure aplikazioari. Zein den esateko:

```
int main(int argc, char** argv)
{
    ...
    /* set the callback functions */
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutSpecialFunc(tekla_berezien_arretarako_funtzioa);
    ...
}
```

Zer egiten duen esateko, keyboard funtzioaren antzeko kodea edukiko du, baina matrizeekin jardun behar du, izan ere aldaketak matrizeen bidez adieraziko baitira:

```
void tekla_berezien_arretarako_funtzioa(int tekla, int x, int y)
{
    switch(tekla)
    {
    case GLUT_KEY_UP:
        // goranzko geziari dagokion matrizea kalkulatu...
        break;
    case GLUT_KEY_RIGHT:
        // eskubiranzko geziari dagokion matrizea...
        break;
    case...
    }
    // kalkulatutako matrizea objektuaren matrizeri bidertu
```

```
    ...
    // emaitza objektuaren matrize modura jaso
    ...
    glutpostredisplay();
}
```

4. Aldaketa desegiteko matrize zerrendako lehenengo elementua kentzea nahikoa da...

5. Amaitzeko, objektuari dagokion matrizea erabili behar da objektua marrazteko, beraz, marrazteko funtzioan, objektuaren matrizea kargatuko dugu MODELVIEW matrizean. Objektuarei gehitu diogun eremu berriaren izena MZptr dela supostzen badugu, eta egitura horretan M eremuan matrizea jasotzen dugula suposatuz:

```
int main(int argc, char *argv[])
{
...
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
/* First , we draw the axes */
draw_axes();
/* Now each of the objects in the list */
while ( aux_obj != 0)
  {
  ...
  /* Objektuaren matrizea kargatu GL_MODELVIEW matrizean */
  glLoadMatrixd(aux_obj->MZptr->M);
  /* Draw the object;
     for each face create a new polygon with the corresponding vertices */
  for (f = 0; f < aux_obj->num_faces; f++)
    {
    glBegin(GL_POLYGON);
    for (v = 0; v < aux_obj->face_table[f].num_vertices; v++)
      {
      v_index = aux_obj->face_table[f].vertex_table[v];
      glVertex3d (aux_obj->vertex_table[v_index].coord.x,
                  aux_obj->vertex_table[v_index].coord.y ,
                  aux_obj->vertex_table[v_index].coord.z );
      }
    glEnd();
    }
  aux_obj = aux_obj->next ;
  }
```