Konputagailu bidezko Grafikoak - 2016/2017 ikasturtea

2. zatia:

2. fasea: Objektuen transformazioak

Egileak: Robin Espinosa eta Julen Ferrero

Aurkibidea

1. Aldaketa #1: Menuko aukerak zabaldu

Azalpena:

Menuko aukerak zabaldu dira. Teklatutik irakurritako teklen ekintzak ere bai.

Aldatutako fitxategia: io.c
Aldatutako metodoa: void keyboard(unsigned char key, int x, int y)

→ case 'm'/case 'M' berria
→ case 'b'/case 'B' berria
→ case 't'/case 'T' berria
→ case 'g'/case 'G' berria
→ case 'l'/case 'L' berria
→ case 26 berria (Ctrl+Z-rako)
→ case '+' zabaldu da (CTRL gabeko aukera gehituta)
→ case '-' zabaldu da (CTRL gabeko aukera gehituta)

2. Aldaketa #2: Objektuen transformaketak teklatutik egiteko aldaketak

Azalpena:

Transformazioak ardatz batean edo bestean egiteko, teklatuko geziak erabiliko dira. Baina gezi hauen pultsaketak irakurtzeko metodo berezi bat implementatu behar izan dugu: keyboardSpecial. Metodo honek GlutSpecialFunc erabiltzen du tekla berezi hauek tratatzeko.

Aldatutako fitxategia: io.c
Metodo berria: voi keyboardSpecial(int key, int x, int y)

→ case GLUT_KEY_LEFT: Mugitu -X; Handitu X; Biratu -Y
→ case GLUT_KEY_UP: Mugitu +Y; Txikitu Y; Biratu +X
→ case GLUT_KEY_RIGHT: Mugitu +X; Txikitu X; Biratu +Y
→ case GLUT_KEY_DOWN: Mugitu -Y; Handitu Y; Biratu -X
→ case GLUT_KEY_PAGE_DOWN: Mugitu +Z; Txikitu Z; Biratu +Z
→ case GLUT_KEY_PAGE_UP: Mugitu -Z; Handitu Z; Biratu -Z

3. Aldaketa #3: Objektuen transformaketak burutzeko klasearen sorrera

Azalpena:

Objektuen transformazioarekin zerikusia duena klase honetan gordeko da.

Sortutako fitxategiak: multiMatrix.c eta multiMatrix.h

→ Klase hau io.c klaseak erabiltzen du.

4. Aldaketa #4: Objektuen transformaketa-mota aukeratzeko aldagaia

Azalpena:

Transformazioaren mota (biraketa, tamaina-aldaketa...) aukera biltegiratzen duen aldagaiaren sorrera eta implementazioa. Adib.: biraketa aukeratuta badago, aldagai honen balioa 'b' izango da.

Aldatutako fitxategia: multiMatrix.c
Sortutako aldagaia: char transf_mota ; io.c-ko keyboard metodoak aldatzen du aldagai hau:
- → case 'm'/case 'M'-an
- → case 'b'/case 'B'-an
- → case 't'/case 'T'-an

5. Aldaketa #5: Objektuen transformaketaren sistema aukeratzeko aldagaia

Azalpena:

Transformazioaren sistema (lokala edo globala) aukera biltegiratzen duen aldagaiaren sorrera eta implementazioa. Era lokalean, aukeratutako objektua transformatuko da; eda globalean, berriz, marraztutako objektu guztiak transformatuko dira.

Aldatutako fitxategia: multiMatrix.c
Sortutako aldagaiak: char transf_helburua ; io.c-ko keyboard metodoak aldatzen du balioa:
- → case 'g'/case 'G'-an
- → case 'l'/case 'L'-an

6. Aldaketa #6: Objektuen transformaketaren ardatza aukeratzeko aldagaia

Azalpena:

Transformazioaren ardatzaren (X, Y edo Z) aukera biltegiratzen duen aldagaiaren sorrera eta implementazioa. Biltegiratutako karakterea erreferentzia-ardatza definitzen du.

Aldatutako fitxategia: multiMatrix.c
Sortutako aldagaia: char transf_ardatza ; io.cko specialKeyboard metodoak aldatzen ditu aldagaiak:
- → case GLUT_KEY_LEFT: Traslazioa: X; Tamaina aldaketa: X; Biraketa: Y
- → case GLUT_KEY_UP: Traslazioa: Y; Tamaina aldaketa: Y; Biraketa: X
- → case GLUT_KEY_RIGHT: Traslazioa: X; Tamaina aldaketa: X; Biraketa: Y
- → case GLUT_KEY_DOWN: Traslazioa: Y; Tamaina aldaketa: Y; Biraketa: X
- → case GLUT_KEY_PAGE_DOWN: Traslazioa: Z; Tamaina aldaketa: Z; Biraketa: Z
- → case GLUT_KEY_PAGE_UP: Traslazioa: Z; Tamaina aldaketa: Z; Biraketa: Z

7. Aldaketa #7: Objektuen transformaketaren norabidea aukeratzeko aldagaia

Azalpena:

Transformazioaren norabidearen (positiboa edo negatiboa) aukera biltegiratzen duen aldagaiaren sorrera eta implementazioa.

Aldatutako fitategia: multiMatri.c
Sortutako aldagaia: char transf_ardatza ; io.c-ko specialKeyboard metodoak aldatzen ditu aldagaiak:
- → case GLUT_KE_LEFT: Traslazioa: -; Tamaina aldaketa: + ; Biraketa: -
- → case GLUT_KE_UP: Traslazioa: +; Tamaina aldaketa: - ; Biraketa: +
- → case GLUT_KE_RIGHT: Traslazioa: +; Tamaina aldaketa: - ; Biraketa: +
- → case GLUT_KE_DOWN: Traslazioa: -; Tamaina aldaketa: + ; Biraketa: -
- → case GLUT_KE_PAGE_DOWN: Traslazioa: +; Tamaina aldaketa: - ; Biraketa: +
- → case GLUT_KE_PAGE_UP: Traslazioa: -; Tamaina aldaketa: + ; Biraketa: -

8. Aldaketa #8: Objektuen transformazio-abiadurak definitzeko aldagai globalen sorrera

Azalpena:

Transformazioak egiteko orduan, transformazio-kopurua definitzeko aldgai globalak sortzea ideia ona da, transformazio “abiadurak” leku bakarretik kontrolatzeko.

--
Aldatutako fitxategia: definitions.h
Balio berriak:

→ #define KG_TAMAN_ABIA 0.1f :// Objektuaren tamaina aldatzeko abiadura
→ #define KG_BIRAK_ABIAD 3.14159265359f/8 // Objektuaren biraketa abiadura
→ #define KG_TRASL_ABIAD 0.1f :// Objektuaren traslazio abiadura

9. Aldaketa #9: Objektuen transformazioak burutzeko jarraitutako pausoak

Azalpena:

Objektuetan transformazioak garatzeko, programaren hainbat klasetan lerro berriak sartu dira. Exekuzioaren ordena jarraituko da objektuen transformazioak azaltzeko.

Objektu bat transformatzeko behar den lehenengo gauza objektu bera da. Eta objektu hori transformazioak jasateko prest egon behar da. Horretarako, objektua definitzeko estrukturan matrize berri bat sortu da: transformazio-matrizea. (1)

Behin hau prestatuta, objektua programan kargatzera pasatuko gara. Objektuaren datuak fitxategitik irakurtzen dira, eta datu hauek objektua sortzeko erabiltzen dira. Hau egin ondoren, transformazio-matrize berria hasierazten zaio, 4x4-ko identitate matrizea. (2)

Hau eginda, objektua transformatzeko prest daukagu. Baina 36 transformaketa aukera ezberdin daude; hauek burutzeko, aldiz, soilik 6 tekla ditugu. Transformazioen helburua (objektu bakarra edo guztiak) eta mota (traslazioa, tamaina-aldaketa edo biraketa) aukeratzeko, aurretik adierazitako teklak erabiltzen dira. Ardatza eta norabidea definitzeko, aldiz, sei kontrol-teklak erabiltzen dira. Tekla bakoitzean, transformazio-motaren arabera, ardatz eta norabide bat definitzen dira, eta ondoren beharrezko metodoa exekutatzen da. (3)

Tekla sakatu dugu, eta nahi den ardatza eta norabidea definitu direla, transformazioa egiteko metodora pasatzen gara. Metodo hauek sortu berri den multiMatrix.c klasean daude. Deitutako metodo honek objektu bakarra edo objektu guztiak transformatzen ditu, aldi berean klase berdinean dagoen *berezko transformazio* metodo bat erabiliz. Objektu guztiak transformatzeko, puntero bat sortu da. Puntero honekin objektu guztiak transformatzen direla kontrolatuko da. Oinarriki, modu globala objektu bakoitzari transformazio metodoa behin aplikatzean datza. (4)

Helburua edozein izanda, *berezko transformazio* metodora sartzen gara. Metodo hauek antzerako egitura dute. Lehenengo, transformazio matrizeak kalkulatzen dira: ardatz bakoitzeko bana, eta norabide bakoitzeko bana; 3 ardatz * 2 norabide: 6 matrize ezberdin. Ondoren, hainbat kontrol egiten dira matrize zuzena aukeratzeko. Lehenengo, switch batekin, ardatza kontrolatzen da, eta ondoren, kasu bakoitzean, norabidea kontrolatzen da. Bukatzeko, objektuak duen lehen sortu berriko transformazio-matrizea metodo honetan definituriko transformazio matrizearekin biderkatzen da. (5) (Biderkaketa burutzeko, metodo berezi bat implementatu behar izan dugu. (6))

Azkenik, objektu transformatua monitorean marrazten da. Horretarako, aukeratutako objektuaren matrizea hasieratu ondoren guk nahi dugun matrizea sartzen dugu glMultMatrixd funtzioa erabiliz. (7)

--

Aldatutako fitxategia #1: definitions.h

-> `struct object3d { ...; Gldouble *matrix; }` (1)

Aldatutako fitxategia #2: load_obj.h

-> (199. lerroa) `object_ptr->matrix = malloc ( sizeof ( GLdouble )*16);`

(ondoren matrizea hasiarazten da identitate-matrize modura); (2)

Aldatutako fitxategia #3: io.c

-> keyBoardSpecial metodoan:

```
-> switch (tekla) { case GORA: (3)
            switch(transf_mota){
            case ‘m’:
                  transf_ardatza = 'Y'; //Ardatza definitzen da
                  transf_norabidea = '+'; //Norabidea definitzen da
                  mugitu(); //Metodoa aplikatzen zaio
                  break;
            ... }
            ... break;
  case...;
  ... }
```

Aldatutako fitxategia #4: multiMatrix.c (eta multiMatrix.h)

-> mugitu/biratu/tAldatu funtzioak (4):

```
      if(transf_helburua=='l'){
            tamainaAldaketa();
        }
        else if(transf_helburua=='g'){
            trans_obj = _first_object;
            while (trans_obj != 0){
                tamainaAldaketa();
                _selected_object = _selected_object->next;
                trans_obj = trans_obj->next;
                if(_selected_object==0) _selected_object= _first_object;
            }
        }
```

-> traslazioa/biraketa/tamainaAldaketa funtzioak (5):

```
   GLdouble  * tamaX = malloc ( sizeof ( GLdouble )*16); //transf. matrizeak
tamaX[0]=1+KG_ABIAD_TAMAN; tamaX[4]=0; tamaX[8] =0; tamaX[12]=0;
tamaX[1]=0;                tamaX[5]=1; tamaX[9] =0; tamaX[13]=0;
tamaX[2]=0;                tamaX[6]=0; tamaX[10]=1; tamaX[14]=0;
tamaX[3]=0;                tamaX[7]=0; tamaX[11]=0; tamaX[15]=1;
...
switch (transf_ardatza){
    case 'X':
        if(transf_norabidea=='+'){
            printf("Tamaina aldaketa: +X\n");
            _selected_object->matrix=mult(_selected_object->matrix,tamaX);
        }
        else{
            printf("Tamaina aldaketa: -X\n");
             _selected_object->matrix=mult(_selected_object->matrix,tamaXn);
        }
        ...
        break;
    case ...
}
```

-> Matrizeak biderkatzeko metodoa: mult(GLdouble *m1, GLdouble *m2) (6):

```
GLdouble *mult(GLdouble *m1, GLdouble *m2) {
    GLdouble *result = malloc(sizeof(GLdouble) * 4 * 4);
    // @formatter:off
result[0+4*0]=m1[0+4*0]*m2[0+4*0]+m1[0+4*1]*m2[1+4*0]+m1[0+4*2]*m2[2+4*0]+m1[0+4*3]*m2[3+4*0];
result[1+4*0]=m1[1+4*0]*m2[0+4*0]+m1[1+4*1]*m2[1+4*0]+m1[1+4*2]*m2[2+4*0]+m1[1+4*3]*m2[3+4*0];
result[2+4*0]=m1[2+4*0]*m2[0+4*0]+m1[2+4*1]*m2[1+4*0]+m1[2+4*2]*m2[2+4*0]+m1[2+4*3]*m2[3+4*0];
result[3+4*0]=m1[3+4*0]*m2[0+4*0]+m1[3+4*1]*m2[1+4*0]+m1[3+4*2]*m2[2+4*0]+m1[3+4*3]*m2[3+4*0];

result[0+4*1]=m1[0+4*0]*m2[0+4*1]+m1[0+4*1]*m2[1+4*1]+m1[0+4*2]*m2[2+4*1]+m1[0+4*3]*m2[3+4*1];
result[1+4*1]=m1[1+4*0]*m2[0+4*1]+m1[1+4*1]*m2[1+4*1]+m1[1+4*2]*m2[2+4*1]+m1[1+4*3]*m2[3+4*1];
result[2+4*1]=m1[2+4*0]*m2[0+4*1]+m1[2+4*1]*m2[1+4*1]+m1[2+4*2]*m2[2+4*1]+m1[2+4*3]*m2[3+4*1];
result[3+4*1]=m1[3+4*0]*m2[0+4*1]+m1[3+4*1]*m2[1+4*1]+m1[3+4*2]*m2[2+4*1]+m1[3+4*3]*m2[3+4*1];

result[0+4*2]=m1[0+4*0]*m2[0+4*2]+m1[0+4*1]*m2[1+4*2]+m1[0+4*2]*m2[2+4*2]+m1[0+4*3]*m2[3+4*2];
result[1+4*2]=m1[1+4*0]*m2[0+4*2]+m1[1+4*1]*m2[1+4*2]+m1[1+4*2]*m2[2+4*2]+m1[1+4*3]*m2[3+4*2];
result[2+4*2]=m1[2+4*0]*m2[0+4*2]+m1[2+4*1]*m2[1+4*2]+m1[2+4*2]*m2[2+4*2]+m1[2+4*3]*m2[3+4*2];
result[3+4*2]=m1[3+4*0]*m2[0+4*2]+m1[3+4*1]*m2[1+4*2]+m1[3+4*2]*m2[2+4*2]+m1[3+4*3]*m2[3+4*2];

result[0+4*3]=m1[0+4*0]*m2[0+4*3]+m1[0+4*1]*m2[1+4*3]+m1[0+4*2]*m2[2+4*3]+m1[0+4*3]*m2[3+4*3];
result[1+4*3]=m1[1+4*0]*m2[0+4*3]+m1[1+4*1]*m2[1+4*3]+m1[1+4*2]*m2[2+4*3]+m1[1+4*3]*m2[3+4*3];
result[2+4*3]=m1[2+4*0]*m2[0+4*3]+m1[2+4*1]*m2[1+4*3]+m1[2+4*2]*m2[2+4*3]+m1[2+4*3]*m2[3+4*3];
result[3+4*3]=m1[3+4*0]*m2[0+4*3]+m1[3+4*1]*m2[1+4*3]+m1[3+4*2]*m2[2+4*3]+m1[3+4*3]*m2[3+4*3];
    // @formatter:on
    return result;
}
```

Aldatutako fitxategia: display.c (7):

-> 111. lerroa: `glMultMatrixd(aux_obj->matrix);`

10. Aldaketa #10: Objektuen transformazioak: traslazioa

Azalpena:

Behin azalduta objektuen trasnformazioen bidea tekla sakatzen denetik objektua pantailaratzen den arte, orain berezko transformazioa azalduko da. Objektuen transformazio bakoitza funtzio batean gordeko da. Objektuen traslazioa kalkulatzeko funtzioa *traslazioa()* da.

Objektua traslatzeko, transformazio matrizea honako hau izango da:

1	0	0	KG_ABIAD_TRASL //X
0	1	0	KG_ABIAD_TRASL //Y
0	0	1	KG_ABIAD_TRASL //Z
0	0	0	1

KG_ABIAD_TRASL aldagaia aurredefinituta dago definitions.h fitxategian. Traslazioa egiteko 6 matrize sortu dira (3 ardatz * 2 norabide). Ardatz bakoitzeko, ardatz horren hizkiarekin dagoen lerroko laugarren zutabea KG_ABIAD_TRASL aldagaiaren balioa izango du; zutabeko beste balioak 0 izango dira (azkenekoa izan ezik, hau beti 1 izango da).

Norabide negatiboa definitzeko, KG_ABIAD_TRASL aldagaiaren balioaren negatiboa kalkulatzen da matrizean bertan.

Aldatutako fitxategia: multiMatrix.c

-> Funtzio berria: traslazioa()

11. Aldaketa #11: Objektuen transformazioak: tamaina aldaketa (ardatz bakarrean)

Azalpena:

Behin azalduta objektuen trasnformazioen bidea tekla sakatzen denetik objektua pantailaratzen den arte, orain berezko transformazioa azalduko da. Objektuen transformazio bakoitza funtzio batean gordeko da. Objektuen tamaina aldaketa kalkulatzeko funtzioa *tamainaAldaketa()* da.

Objektuaren tamaina ardatz batean aldatzeko, transformazio matrizea honako hau izango da:

1+KG_ABIAD_TAMAN	0	0	1 //X
0	1+KG_ABIAD_TAMAN	0	1 //Y
0	0	1+KG_ABIAD_TAMAN	1 //Z
0	0	0	1

KG_ABIAD_TAMAN aldagaia aurredefinituta dago definitions.h fitxategian. Tamaina aldaketa egiteko 6 matrize sortu dira (3 ardatz * 2 norabide). Ardatz bakoitzeko, ardatz horren hizkiarekin dagoen lerroaren diagonal zutabea 1+KG_ABIAD_TAMAN aldagaiaren balioa izango du; zutabeko beste balioak 0 izango dira (azkenekoa izan ezik, hau beti 1 izango da).

Norabide negatiboa definitzeko, 1-KG_ABIAD_TAMAN egingo da.

Aldatutako fitxategia: multiMatrix.c

-> Funtzio berria: tamainaAldaketa()

12. Aldaketa #12: Objektuen transformazioak: tamaina handitzea (ardatz guztietan)

Azalpena:

Behin azalduta objektuen trasnformazioen bidea tekla sakatzen denetik objektua pantailaratzen den arte, orain berezko transformazioa azalduko da. Objektuen transformazio bakoitza funtzio batean gordeko da. Objektuen tamaina aldaketa kalkulatzeko funtzioa *guztiaHanditu()* da.

Objektuaren tamaina orokorrean handitzeko, transformazio matrizea honako hau izango da:

```
1+KG_ABIAD_TAMAN          0                0                1 //X
0          1+KG_ABIAD_TAMAN       0                1  //Y
0                    0       1+KG_ABIAD_TAMAN       1  //Z
0                    0                0                1
```

KG_ABIAD_TAMAN aldagaia aurredefinituta dago definitions.h fitxategian. Tamaina aldaketa egiteko matrize bakarra sortu da.

Matrizearen diagonaleko balio guztiak 1+KG_ABIAD_TAMAN aldagaiaren balioa izango dute (azkenekoa izan ezik, hau beti 1 izango da).

Aldatutako fitxategia: multiMatrix.c

-> Funtzio berria: guztiaHanditu()

13. Aldaketa #13: Objektuen transformazioak: tamaina txikitzea (ardatz guztietan)

Azalpena:

Behin azalduta objektuen trasnformazioen bidea tekla sakatzen denetik objektua pantailaratzen den arte, orain berezko transformazioa azalduko da. Objektuen transformazio bakoitza funtzio batean gordeko da. Objektuen tamaina aldaketa kalkulatzeko funtzioa *guztiaTxikitu()* da.

Objektuaren tamaina orokorrean txikitzeko, transformazio matrizea honako hau izango da:

```
1-KG_ABIAD_TAMAN          0                0                1 //X
        0        1-KG_ABIAD_TAMAN         0                1 //Y
        0                 0        1-KG_ABIAD_TAMAN         1 //Z
        0                 0                0                1
```

KG_ABIAD_TAMAN aldagaia aurredefinituta dago definitions.h fitxategian. Tamaina aldaketa egiteko matrize bakarra sortu da.

Matrizearen diagonaleko balio guztiak 1-KG_ABIAD_TAMAN aldagaiaren balioa izango dute (azkenekoa izan ezik, hau beti 1 izango da).

Aldatutako fitxategia: multiMatrix.c

-> Funtzio berria: guztiaHanditu()

14. Aldaketa #14: Objektuen transformazioak: biraketa

Azalpena:

Behin azalduta objektuen trasnformazioen bidea tekla sakatzen denetik objektua pantailaratzen den arte, orain berezko transformazioa azalduko da. Objektuen transformazio bakoitza funtzio batean gordeko da. Objektuen biraketa kalkulatzeko funtzioa *biraketa()* da.

X ardatzarekiko biraketa-marizea:

1	0	0	0
0	cos(KG_ABIAD_BIRAK)	sin(KG_ABIAD_BIRAK)*(-1)	0
0	sin(KG_ABIAD_BIRAK)	cos(KG_ABIAD_BIRAK)	0
0	0	0	1

Y ardatzarekiko biraketa-matrizea:

cos(KG_ABIAD_BIRAK)	0	sin(KG_ABIAD_BIRAK)	0
0	1	0	0
sin(KG_ABIAD_BIRAK)*(-1)	0	cos(KG_ABIAD_BIRAK)	0
0	0	0	1

Z ardatzarekiko biraketa-matrizea:

cos(KG_ABIAD_BIRAK)	sin(KG_ABIAD_BIRAK)*(-1)	0	0
sin(KG_ABIAD_BIRAK)	cos(KG_ABIAD_BIRAK)	0	0
0	0	1	0
0	0	0	1

KG_ABIAD_BIRAK aldagaia aurredefinituta dago definitions.h fitxategian. Tamaina aldaketa egiteko 6 matrize sortu dira (3 ardatz * 2 norabide). Norabide negatiboa definitzeko, sin() funtzioaren negatiboa kalkulatuko da (funtzioa aurretik negatiboa bada, positibo bihurtuko da).

Aldatutako fitxategia: multiMatrix.c

-> Funtzio berria: biraketa()

15. Aldaketa #15: Objektuak jasandako aldaketak desegiteko aukera

Azalpena:

Objektuak jasandako aldaketak desegiteko, hainbat aldaketa egin ditugu programaren hainbat klaseetan.

Objektu baten aldaketak desegiteko, objektua aldatuta egon beharko da eta objektuak jazarritako aldaketen informazioa izan behar dugu. Horretarako, objektu bakoitzaren aldaketak “pila” batean gordetzen dira. Objektu bakoitzaren pila egiteko estruktura bat sortu da “aldaketaPila” deitzen dena, non aldaketa egin ostean lortzen dugun matrizearen informazioa gordetzen den. Objektua definitzeko, estrukturan pila bat deklaratu dugu, baita aldaketen kontagailu bat ere. (1)

Behin hau prestatuta, objektua programan kargatzera pasatuko gara. Objektuaren datuak fitxategitik irakurtzen dira eta datu hauek objektua sortzeko erabiltzen dira. Hau egin ondoren, transformazio-matrize berria hasierazten zaio, 4x4-ko identitate matrize bat, zein pilaren lehenengo elementua izango da. Azkenik, objektu horren aldaketa kontagailu hutsera hasierazten da. (2)

Hau eginda, objektu bati aldaketa bat ezartzen zaionean, aldaketa hori egiterakoan sortzen den matrize berria sartzen da objektu horren pilan eta aldaketen kontagailua batean gehituko da. Horretarako pilanGehitu(GLdouble *matrix_aux) erabiliko da (3). Funtzio hau aldaketak egiten diren funtzioetan aipatuko da. (4)

Informazio hau izanda, aldaketak desegiteko prest gaude. Funtzio hau egiteko Ctrl + Z teklak erabiltzen dira. (5) Tekla konbinazio hau sakatzerakoan, aldaketen kontagailua zero baino handiagoa bada, hautatutako objekturen azken aldaketa desegiten da eta objektuaren pilaren goiko matrizea ezabatuko da. Aldaketen kontagailua ere -1ra aldatzen da. Horretarako, aldaketaDesegin() funtzioa erabiliko da. (6)

--

Aldatutako fitxategia #1: definitions.h (1)

```
struct aldaketaPila{GLdouble * matrix;struct aldaketaPila *next;};
typedef struct aldaketaPila pila;
struct object3d { ...; pila * aldaketaPila; GLint num_aldaketak; }
```

Aldatutako fitxategia #2: load_obj.c (2)

(208. lerrotik 211. lerrora)

```
pila * pilaBerria = (pila *)malloc(sizeof(pila));
pilaBerria->matrix = object_ptr->matrix;
object_ptr->aldaketaPila = pilaBerria;
object_ptr->num_aldaketak=0;
```

Aldatutako fitxategia #3: multiMatrix.c (3)

```
void pilanGehitu(GLdouble *matrix_aux){
    pila *pila_aux = (pila *) malloc(sizeof(pila));
    pila_aux->matrix= matrix_aux;
    pila_aux->next = _selected_object->aldaketaPila;
    _selected_object->aldaketaPila = pila_aux;
    _selected_object->num_aldaketak+=1;
}
```

(146,158,170,227,239,251,307,319,331, 406 eta 421. lerroetan)
`pilanGehitu(_selected_object->matrix);`(4)

```
void aldaketaDesegin(){ (6)
    if(_selected_object->num_aldaketak>0){
        pila *first_matrix;
        first_matrix = _selected_object->aldaketaPila->next;
        _selected_object->aldaketaPila = 0;
        _selected_object->aldaketaPila = first_matrix;
        _selected_object->matrix = first_matrix->matrix;
        _selected_object->num_aldaketak-=1;
        printf("Aldaketak desegin dira\n");}
    else{
        printf("Ezin dira aldaketa gehiagorik desegin\n");}
}//aldaketaDesegin
```

Aldatutako fitxategia #4: io.c (5)

```
void keyboard(unsigned char key, int x, int y) {
      switch (key) {
            case 26: /* <CTRL + Z*/
                        printf("Aldaketak desegin\n");
                        aldaketaDesegin();
                        break;
      }
}
```

16. Oharrak

→ Programa konpilatzeko:

```
gcc -o KbGprograma *.c -lGL -lGLU -lglut
```

→ Programa exekutatzeko:

```
./KbGprograma
```