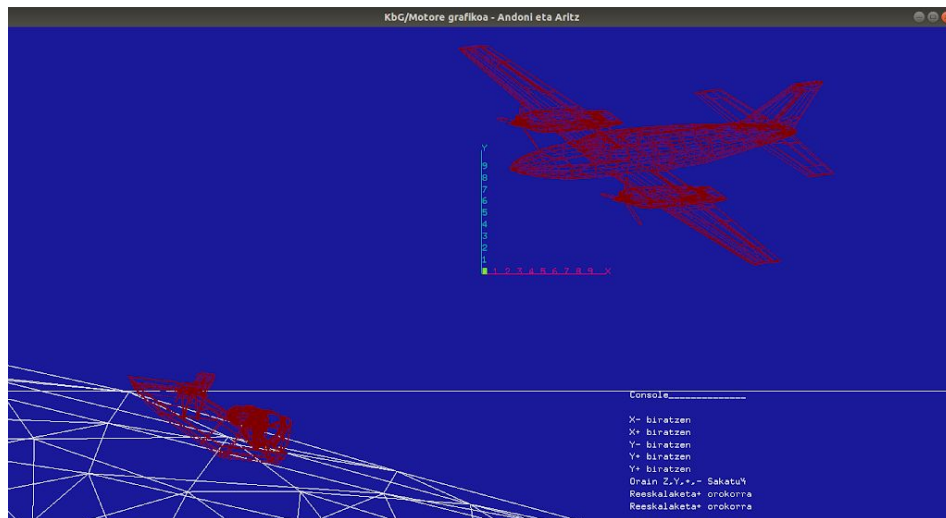


Motore Grafikoa

“Andoni Aranguren eta Aritz Ibarra”



Konputagailu bidezko grafikoak
EHU-UPV, Gipuzkoa

Sarrera	2
1. Fasea: Kodearen konponketa	2
Planoaren kolorea aldatu	2
Deskribapenak:	2
Objektuen kolorea aldatu	3
Deskribapenak:	3
Ardatzen kolorea aldatu	4
Deskribapenak:	4
2. Fasea: Aldaketak	5
Objektuen aldaketak	5
Stack.c eta stack struct-a	5
Traslazio matrizeak:	14
Biraketa matrizeak:	15
Tamaina aldaketa matrizea:	15
Islapen matrizea:	16
Input eta Output	17
Tekla normalak	17
Deskribapenak:	17
Aldaketei bideratutako teklak	19
Deskribapenak:	19
Kontsola	20

Sarrera

Motore grafiko sinple bat eman zitzaigun, motore grafiko honek aukera batzuk zituen baina baita arazo batzuk konpondu beharrekoak. Gure helburua motore grafiko honen funtzionamendu egokia izango da.

Lehenik eta behin programa konpilatzea eta exekutatzeari eskatu zitzaigun. Honetarako hurrengoko komandoa erabili genuen “Iturburu-kodea” direktorioan (`gcc -o KbGprograma *.c -lGL -lglut`). Honekin KbGprograma lortuko genuke eta hau exekutatzekoan motore grafikoa martxan jarriko genukeen.

1. Fasea: Kodearen konponketa

Planoaren kolorea aldatu

Planoaren kolorea eraldatzeko “definitions.h” fitxategian hurrengoko aldagaiak eraldatu beharko ditugu:

Definitions.h

(...)

```
#define KG_COL_BACK_R      0.1f
#define KG_COL_BACK_G      0.1f
#define KG_COL_BACK_B      0.6f
#define KG_COL_BACK_A      1.00f
```

(...)

Deskribapenak:

KG_COL_BACK_R

Kolore gorriaren kantitatearen portzentaia adierazten du.

KG_COL_BACK_G

Kolore berdearen kantitatearen portzentaia adierazten du.

KG_COL_BACK_B

Kolore urdinaren kantitatearen portzentaia adierazten du.

KG_COL_BACK_A

Atzeko planoaren gardentasun koefizientea adierazten du.

Objektuen kolorea aldatu

Objektuaren kolorea eraldatzeko “definitions.h” fitxategian hurrengoko aldagaiak eraldatu beharko ditugu:

```
Definitions.h

(...)

#define KG_COL_SELECTED_R      1.00f
#define KG_COL_SELECTED_G      1.00f
#define KG_COL_SELECTED_B      1.00f

(...)
```

Deskribapenak:

KG_COL_SELECTED_R

Kolore gorriaren kantitatearen portzentaia adierazten du.

KG_COL_SELECTED_G

Kolore berdearen kantitatearen portzentaia adierazten du.

KG_COL_SELECTED_B

Kolore urdinaren kantitatearen portzentaia adierazten du.

Ardatzen kolorea aldatu

Ardatzen kolorea eraldatzeko “definitions.h” fitxategian hurrengoko aldagaiak eraldatu beharko ditugu:

Definitions.h	
(...)	
#define KG_COL_X_AXIS_R	0.9f
#define KG_COL_X_AXIS_G	0.0f
#define KG_COL_X_AXIS_B	0.4f
#define KG_COL_Y_AXIS_R	0.0f
#define KG_COL_Y_AXIS_G	0.8f
#define KG_COL_Y_AXIS_B	0.65f
#define KG_COL_Z_AXIS_R	0.55f
#define KG_COL_Z_AXIS_G	0.95f
#define KG_COL_Z_AXIS_B	0.0f
(...)	

Deskribapenak:

KG_COL_X_AXIS_R

Kolore gorriaren kantitatearen portzentaia adierazten du X ardatzerako.

KG_COL_X_AXIS_G

Kolore berdearen kantitatearen portzentaia adierazten du X ardatzerako.

KG_COL_X_AXIS_B

Kolore urdinaren kantitatearen portzentaia adierazten du X ardatzerako.

KG_COL_Y_AXIS_R

Kolore gorriaren kantitatearen portzentaia adierazten du Y ardatzerako.

KG_COL_Y_AXIS_G

Kolore berdearen kantitatearen portzentaia adierazten du Y ardatzerako.

KG_COL_Y_AXIS_B

Kolore urdinaren kantitatearen portzentaia adierazten du Y ardatzerako.

2. Fasea: Aldaketak

Fase honetan gure Motore grafikoari urrengokofuntzionalitate berri batzuk gehitzea eskatu zaigu:

- Objektuen **translazioak, biraketak eta tamaina aldaketak** inplementatu.
- Aldaketak erreferentzi sistema lokaleak zein globaleak egiteko aukera eman.
- Aldaketak desegiteko aukera eman.

Baita hautazko batzuk proposatu egin zitzaizkigun:

- Aldaketak berregiteko aukera
- Aldaketa berriak inplementatu (islapena, adibidez)

Gure kasuan derrigorrezko atazak burutzea lortu egin dugu eta hautazkoak baita inplementatu egin ditugu.

Objektuen aldaketak

Stack.c eta stack struct-a

Objektuen gaineko transformazioak ahalbidetzeko transformazio matrizeen erabilpenaren beharra dago. Gure kasuan, transformazio matrizeen funtzionamendu egokia bermatzeko, hurrengoko egitura sortu dugu:

- Alde batetik stack struct-a daukagu. Honetan transformazio matrizeen array bat izango dugu. Hasierako transformazio matrizea identitate matrizea izango da, era honetan aldaketarik egin gabe objektua bueltatuko dugu. Hau GLdouble motatakoa izango da eta KG_STACK_MAX_SIZE aldagai globalak definitu egingo du (1024). Era honetan objektuari egiten zaizkion aldaketa guztiak kontutan eta gordeta izango ditugu. Baita erabiliko ditugu bi iteratzaile
 - Lehenengoa “*selected*” izango da eta adierazi egingo digu zein den momentuan daukagun objektuaren eraldaketa aukeraturik.
 - Bestalde “*upto*” adieraziko digu gure pilan zenbat eraldaketa ditugun gordeta.

definitions.h

(...)

```
typedef struct console console;
```

```
struct stack {  
    GLdouble *matrizeak[KG_STACK_MAX_SIZE];  
    int selected;  
    int upto;  
};
```

```
typedef struct stack stack;
```

```
/*  
*****
```

```
 * Structure to store a      *
```

```
 * pile of 3D objects      *
```

```
*****  
*/
```

```
struct object3d{  
    char *fitx_izena;          /* objektuaren fitxategiaren izena*/  
    GLint num_vertices;        /* number of vertices in the object*/  
    vertex *vertex_table;      /* table of vertices */  
    GLint num_faces;           /* number of faces in the object */  
    face *face_table;          /* table of faces */  
    point3 min;                 /* coordinates' lower bounds */  
    point3 max;                 /* coordinates' bigger bounds */  
    struct object3d *next;      /* next element in the pile of objects */  
    stack *s;  
};
```

```
typedef struct object3d object3d;
```

(...)

Orain stack.c zatika azaltzen joango gara uler dezagun zelan funtzionatzen duen. Hemendik aurrera azalduko den kode guztia pilaren funtzionamendurako inplementaturik dago.

stack.c

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <math.h>
#include <GL/glut.h>
#include "definitions.h"
#include "console.h"

(...)
```

Kode zati honetan aurrerago beharko ditugun metodoen edo aldagaien inportazioak erabiliko ditugu.

stack.c

```
(...)
stack* stack_initialization(){
    stack *s =(stack*) malloc(sizeof(stack));
    s->selected = 0;
    s->matrizeak[KG_STACK_MAX_SIZE];
    return s;
};

(...)
```

Stack-a hasieratzeko alokatu egingo dugu bere buruak behar izango duen memoria espazioa ((stack*) malloc(sizeof(stack))). Ostean selected aldagaia hasieratu egingo dugu 0 baliora, azken finean oraindik ez daukagu objekturik sartuta gure pilan. Baita hasieratu egingo dugu “matrizeak” aldagaia bere barnean aldaketa guztiak gordeko ditugulako.

Matrizeak biderkatzeko eta gure objektuari aldaketak aplikatzeko hurrengoko funtzioa sortu behar izan genuen.

stack.c

```
(...)  
GLdouble* matrix_multiplication(GLdouble* m1, GLdouble* m2){  
    GLdouble *emaitza=0;  
    emaitza =(GLdouble*)malloc(sizeof(GLdouble)*16);  
    for(int i=0;i<4;i++){  
        for(int j=0;j<4;j++){  
            float bat = 0.0;  
            for (int k = 0; k < 4; k++){  
                bat = bat + m1[i * 4 + k] * m2[k * 4 +j];  
            }  
            emaitza[i * 4 +j] = bat;  
        }  
    }  
    return emaitza;  
}  
(...)
```

Kodean ikus daitekeenez lehenik eta behin emaitza aldagaia hasieratu egingo dugu. Ostean memorian tokia alokatu egingo diogu. Behin hau egin dugula hasiko gara matrizeen arteko biderketa burutzen. Lehenik eta behin elementu bakoitza matrizeko beste elementuekin biderkatuta lortuko dugu gure objektua nola gelditu den eraldaketa ondoren. Kontuan izanda m1 orain arte gure objektuari aplikaturiko aldaketak direla eta m2 aldaketa berria izango dela aldaketa lokalean gaudenean eta aldaketa globalean aldrebes izango da.

Orain Matrizeak pilan sartzeko funtzioa sortuko dugu.

stack.c

```
(...)  
void matrix_into_stack(GLdouble* m1, GLdouble* m2, stack* s, char  
aldaketa){  
    if (s->selected < KG_STACK_MAX_SIZE-1) {
```

```

    if (aldaketa=='g'){
        s->matrizeak[++s->selected] = matrix_multiplication(m1,m2);
    }
    else if (aldaketa=='l'){
        s->matrizeak[++s->selected] = matrix_multiplication(m2,m1);
    }
    ++s->upto;
}
else {
    printf("Stack-aren kapazitate maximora heldu zara\n");
}

(...)

```

Lehenik eta behin begiratuko dugu pilaren maximora ez garela heldu. Horrela bada begiratu egingo dugu aldaketa globalean edo aldaketa lokalean gauden. Behin hori dakigula “*matrix_multiplication*” eginda lortuko dugu gure objektuaren eraldaketa. Behin jada stack-ean sartu dugula “*upto*” aldagaia haunditu egingo dugu.

Baina maximora heldu bagara abisatu egingo dugu, jada ezin dugula gehitu eraldaketa berririk.

Gero stack-etik lehenengo eraldaketa lortzeko hurrengoko funtzioa sortu behar izan dugu.

stack.c

```

(...)
GLdouble* peak(stack* s){
    if (s->selected == 0){
        GLdouble* ident = malloc(sizeof(GLdouble)*16);
        ident[0] = ident[5] = ident[10] = ident[15] = 1.;
        ident[1] = ident[2] = ident[3] = ident[4] = ident[6] = ident[7] =
        ident[8] = ident[9] = ident[11] = ident[12] = ident[13] = ident[14] = 0.;
        return ident;
    }
    return s->matrizeak[s->selected];
}

```

(...)

Funtzio honetan ikus dezakegu nola objekturik ez badaukagu orduan identitate matrizea bueltatuko dugu. Objektua edukita pilan sartutako azken eraldaketa lortuko dugu.

Metodo bat behar dugu pilan atzera joateko, hau da aurreko eraldaketara heltzeko.

stack.c

```
(...)  
void pop(stack* s){  
    if (s->selected > 0){  
        s->matrizeak[--s->selected];  
    }  
}  
(...)
```

Kodean ikusi daitekeenez “select” aldagaiari atzera joatea egingo dugu, era honetan lortuko dugu aurreko eraldaketara bueltatzea.

Baita metodo bat beharko dugu pilan aurrera joateko.

stack.c

```
(...)  
void redo(stack* s){  
    if(s->selected < s->upto){  
        s->matrizeak[++s->selected];  
    }  
}  
(...)
```

Kasu honetan begiratuko dugu ditugun eraldaketen maximora heldu ez bagara eta horrela bada aurrera egingo dugu eraldaketen pilan.

Metodo bat sortu dugu baita kontsolan matrizeak inprimatzeko, era honetan arazoren bat izatekotan ikusiko litzateke zer nolako matrizea geneukan.

stack.c

```
(...)  
void inprimatu_matrizea(GLdouble *m){  
    for(int i=0; i<4; i++){  
        printf("%f %f %f %f\n",m[i],m[i+4],m[i+8],m[i+12]);  
    }  
    printf("\n");  
}  
(...)
```

Metodo honetan era iteratibo batean errenkada inprimatu egingo dugu matrizea.

Orain stack-aren atalik garrantzitsuena azaldu egingo dugu. Eraldaketa matrizen sorrerari dagokion funtzioa.

stack.c

```
(...)  
void stack_add(stack *s, int code, char egoera, char aldaketa){  
    GLdouble* m= malloc(sizeof(GLdouble)*16);  
    switch (code) {  
        case 101: /*UP*/  
            if (egoera == 'm'){  
                console_add("Y+ mugitzen");  
                (...) /*Transformazioa matrizea definitzen da*/  
            }  
            else if (egoera == 'b'){  
                console_add("X+ biratzen");  
                (...) /*Transformazioa matrizea definitzen da*/  
            }  
    }
```

```

        else if (egoera == 't'){
            console_add("Y- reeskalatzen");
            (...) /*Transformazioa matrizea definitzen da*/
        }
        else if(egoera == 'o'){
            console_add("Y islatzen");
            (...) /*Transformazioa matrizea definitzen da*/
        }
        break;
    case 103: /*DOWN*/ (...) /*Goiko kasuaren antzekoa*/
        break;
    case 100: /*LEFT*/ (...) /*Goiko kasuaren antzekoa*/
        break;
    case 102: /*RIGHT*/ (...) /*Goiko kasuaren antzekoa*/
        break;
    case 104: /*REpag*/ (...) /*Goiko kasuaren antzekoa*/
        break;
    case 105: /*AVpag*/ (...) /*Goiko kasuaren antzekoa*/
        break;
    case '+': (...) /*Goiko kasuaren antzekoa kasu bakarrarekin*/
        break;
    case '-': (...) /*Goiko kasuaren antzekoa kasu bakarrarekin*/
        break;
    case 114:
        console_add("Orain Z,Y,+, - Sakatu");
        break;
    default:
        console_add("Stack-era sartzeko komandoa ez da identifikatu");
        break;
}
}
(...)

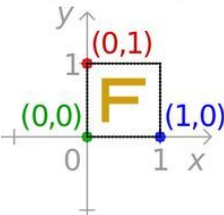
```

Lehenik eta behin memorian tokia gorde egingo diogu eraldaketa matrizea gordetzeko. Ostean switch bat sortuko dugu kasu desberdinak aztertzeko. Eta haien barnean ondoren azalduko den kodea sartuko dugu. Kontuan izanda kasu bakoitzerako desberdina izango direla balio batzuk. Hau egoeraren arabera aldatu egingo dira.

stack.c

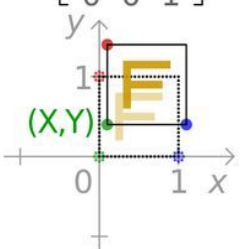
```
(...)  
switch (code) {  
    case 101: /*UP*/  
        ###  
    case 103: /*DOWN*/  
        ###  
    case 100: /*LEFT*/  
        ###  
    case 102: /*RIGHT*/  
        ###  
    case 104: /*REpag*/  
        ###  
    case 105: /*AVpag*/  
        ###  
    case '+':  
        ###  
    case '-':  
        ###  
}  
(...)
```

Jartzen duen tokietan eraldaketa matrizea eraldatu egingo dugu eta ostean kasu berezi horri dagokion eraldaketak egingo ditugu matrize horretan. Era honetan lortu egingo dugu.

Matrizea	2D-n adibidea
$\begin{pmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{pmatrix}$	<p>No change</p> $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ 

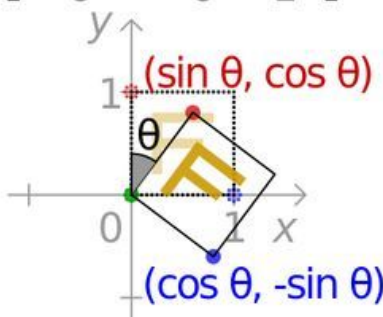
Traslazio matrizeak:

Eskumako zutabeen balioak banaka sartzen dira matrize ezberdinetan. Eraldaketa matrizearen posizio hauek objektuarekiko biderkatuta mugitu egingo dugu. Aldaketa Globala denean, ardatzei dagokionez mugituko da. Aldiz, aldaketa lokala denean bere ardatzei dagokionez mugituko da.

Matrizea	2D-n adibidea
$\begin{pmatrix} I & O & O & \begin{matrix} +- X \\ \text{ardatzeko} \\ \text{aldaketa} \end{matrix} \\ O & I & O & \begin{matrix} +- Y \\ \text{ardatzeko} \\ \text{aldaketa} \end{matrix} \\ O & O & I & \begin{matrix} +- Z \\ \text{ardatzeko} \\ \text{aldaketa} \end{matrix} \\ O & O & O & I \end{pmatrix}$	<p>Translate</p> $\begin{bmatrix} 1 & 0 & X \\ 0 & 1 & Y \\ 0 & 0 & 1 \end{bmatrix}$ 

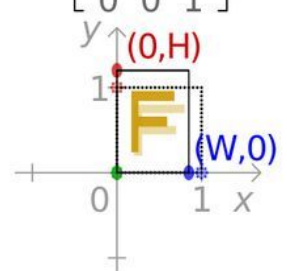
Biraketa matrizeak:

Rxy , Ryz, Rxz matrizeak erabilia lortuko dugu objektua biratzea, aldaketa globala daukagunean ardatzarekiko biratuko da. Aldiz, lokala denean bere ardatzei dagokionez biratuko da.

Matrizeak	2D-n adibidea
<p style="text-align: center;">Rotation</p> $\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{X axis}$ $\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{Y axis}$ $\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{Z axis}$ <p>New points rotation matrix old points</p>	<p>Rotate about origin</p> $\begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$ 

Tamaina aldaketa matrizea:

Objektu baten tamaina aldaketa egiteko diagonalean kokatzen diren koordinatueta aldaketak egin behar dira.

Matrizeak	2D-n adibidea
$H_s(\vec{S}) = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ <p>Where: S is a 3D Scaling Vector</p>	<p>Scale about origin</p> $\begin{bmatrix} W & 0 & 0 \\ 0 & H & 0 \\ 0 & 0 & 1 \end{bmatrix}$ 

Islapen matrizea:

Islapenari dagokionez diagonalean -1 balioa jarritz lortuko dugu islapena. Aldaketa globalean ardatzarekiko egingo da islapena. Aldiz, aldaketa lokalean gaudenean bere buruarekiko islatuko da.

Matrizeak	$H_s(\vec{S}) = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ <p>Non S_x, S_y, S_z -1 balioa hartzen duten islapena egiteko</p>
2D-n adibidea	<div> <div> <p>Reflect about origin</p> $\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ </div> <div> <p>Reflect about x-axis</p> $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ </div> <div> <p>Reflect about y-axis</p> $\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ </div> </div>

Input eta Output

Tekla normalak

Gure programaren funtzionalitate guztiak ahalbidetzeko "io.c" fitxategian tekleen portaera orokorra kodetu dugu. Tekla normal nagusiak hurrengoak dira:

io.c

```
void keyboard(unsigned char key, int x, int y) {  
    (...)  
  
    console_add("<?> Laguntza hau bistaratu");  
    console_add("<ESC> Programatik irten ");  
    console_add("<F> Objektua bat kargatu");  
    console_add("<I> Hautatutako objektuaren informazioa");  
    console_add("<TAB> Kargaturiko objektuen artean bat  
hautatu");  
    console_add("<DEL> Hautatutako objektua ezabatu");  
    console_add("<CTRL + -> Bistaratzeko eremua handitu");  
    console_add("<CTRL + +> Bistaratzeko eremua txikitu");  
  
    (...)
```

Deskribapenak:

<?> tekla

Informazioa bistaratzen duen tekla da.

→ Funtzio bati deitzen dio informazioa printeatzeko. Gure kasuan kontsolan.

<ESC> tekla

Programa ixten duen tekla.

→ exit(0) egiten du.

<F> tekla

Objektu berriak kargatzeko tekla. Erabiltzaileak .obj baten helbidea sartu behar du.

→ Terminalean helbide bat idaztea beharrezkoa da.

→ "load_obj.c" fitxategia erabiltzen dugu objektuak kargatzeko.

→ Objektuen izena, erpinak ... etb. gordetzen dira object3d struct batean.

→ Objektuaren helbidea okerra bada, programak ez du ezer kargatuko.

<I> tekla

Hautatutako objektuaren informazioa bueltatzen du.

→ _selected_object pointer-reko objektuaren izena bueltatzen du kontsolan.

<TAB> tekla

Hautatutako objektua aldatzen da, listan hurrengoa hartuz.

→ `_selected_object` pointer-reko objektuaren `.next` hartzen du.

** tekla**

Hautatutako objektua ezabatzen du.

→ `_selected_object` pointer-reko objektuaren ezabatzen du.

→ Ez badago objekturik ezabatzeko, ez du ezer egingo.

<CTRL +> tekla

Munduko bistaratze eremua handitzen du.

→ Mundua bistaratzeko eremuaren balioak handitzen ditu.

→ Ez bada CTRL sakatzen, objektuen tamaina aldatuko da.

<CTRL -> tekla

Munduko bistaratze eremua txikitzen du.

→ Mundua bistaratzeko eremuaren balioak txikitzen ditu.

→ Ez bada CTRL sakatzen, objektuen tamaina aldatuko da.

Aldaketei bideratutako tekla

Gure programan objektuen gaineko aldaketak ahalbidetzeko “io.c” fitxategian tekla berezien kudeaketa egin da. Tekla kodea honelako itxura du:

io.c

```
void keyboardSpecial(int key, int x, int y){
(...)
case 101:
case 100:
(...)
case 114:
if((egoera=='b')||(egoera=='t')||      (egoera=='m')      ||
(egoera=='o')){
    if (_selected_object != NULL){
        stack_add(_selected_object->s,key,egoera,aldaketa); }
    }
else if (glutGetModifiers() != GLUT_ACTIVE_CTRL){
    console_add("Aldaketa mota aukeratu (B/T/M/O)"); }
(...)
}
```

Goiko kode zatian ikusi ahal denez, tekla objektuetan duten eragina “stack_add” funtzioaren bitartez kudeatzen da. Hau honela egiten da “stack.c” fitxategia matrizeen transformazio

Deskribapenak:

<G> tekla

Aldaketak globalki egitea.

→Aldaketak munduaren ardatzen gainean egingo dira.

<L> tekla

Aldaketak lokalki egitea.

→Aldaketak objektuaren ardatzen gainean egingo dira.

 tekla

Biraketaren transformazioa aukeratzeko tekla.

→ Hautatutako objektua birarazten du gero norabide tekla sakatuta.

<T> tekla

Tamaina eraldatzeko transformazioa aukeratzeko tekla.

→ Hautatutako objektua reeskalatzen du gero norabide tekla sakatuta.

<M> tekla

Translazio transformazioa aukeratzeko tekla.

→ Hautatutako objektua mugitzen du gero norabide teklak sakatuta.

<Norabide> teklak

Aukeratutako transformazioa X,Y ardatzetan eragiteko erabiltzen diren teklak.

→ Norabide teklak aldaketak zein ardatzen gainean eta zein norabidetan egingo diren ezartzen dute.

<RePag eta AvPag> teklak

Aukeratutako transformazioa Z ardatzean eragiteko erabiltzen diren teklak.

→ Bi tekla hauek Z ardatzean aldaketak zein norabidetan egingo diren ezartzen dute.

<- eta +> teklak

Hautatutako objektuaren tamaina aldatzen du.

→ X,Y eta Z ardatzetan minusak objektua txikitzen du, eta plusak handitu.

Kontsola

Programan egindako operazioak errazago jarraitzeko, programaren barruko UI-an (User-Interface-an) kontsola bat sortu dugu. Kontsola hau, negozio logikan String-en array bat baino ez da. Baina mezuak aurrera eta atxera begiratzeko aukera ematen digu. Honela, gure UI-an egin ditugun azkeneko operazioak edo eskatutako informazioa ikusteko ahalmena ematen digu.

Geroago honetan idazteko ahalmena ematea inplementatuko asmoa dugu. Baina orain arte, programak erabiltzaileari testu bat sartzea eskatzekotan, kanpoko terminalera joan beharko da.

Kontsolaren kokapena leihoaren tamainarekin kalkulatzeko da, baina balioak hardcoded-tuta daude oraintze bertan. Geroago mezu kopuru ezberdinak, testua sartzeko ahalmena... eta orokorrean kontsola txukunago inplementatuko da. Baina horretarako "console.c" eta console struct bat sortu dugu. Guztia kodearen eremu ezberdinetan ondo bananduta izateko.

Hemen ez da honen inguruko azalpenik emango, oraindik ez baitago amaituta. Halere, "stack.c" eta stack struct-arenantzeko funtzionamendua daukala esan ahal dugu. Mezua gehitu, mezua aurrera eta atzera eta honelako funtzioak erabiltzen baititu.