

Nombre del alumno:

Esquivel Garza Josué Israel

Matrícula:

281876

Clave de grupo:

280702

Materia:

Graficación por computadora

Maestro:

Omar Rodríguez Gonzales

Proyecto: "Atrápala bro"

Fecha:

13 de junio de 2021

ÍNDICE

Introducción:	4
Desarrollo:	5
Lector Obj	5
Diagrama de flujo del lector:	5
Diagrama de flujo del proyecto:.....	6
Object:	7
HPP:	8
CPP:	9
Vertex:	11
HPP:	11
CPP:	11
Face:.....	12
HPP:	12
CPP:	13
Transform.....	14
HPP:	14
CPP:	14
Hbox:	16
Persona:.....	18
HPP.....	18
CCP:	19
Palomita	20
HPP:	20
CPP:	21
Monkey:.....	23
HPP:	23
CPP.....	24
Main:.....	26
Key_callback:	26
Colisión:	27
OpenGL.....	31
Desarrollo específico (funciones)	33

Tiro parabólico: Curvas de Bézier.....	33
Colisión:	34
Problemas:.....	36
Conclusiones:.....	37
Referencias:	37

Introducción:

El proyecto consiste en una especie de minijuego que representa una escena de una serie popular llamada: “El laboratorio de Dexter” en la cual el villano le arroja palomitas a Monkey y él come una sin querer y a este no le gusta, todo parte de una distracción para intentar derrotar a Monkey.

Mi proyecto representa a el villano (Persona) arrojando la palomita y el objetivo de Monkey es cacharla.



Esto nos ayuda a contextualizar de manera mas formal los conceptos que veníamos viendo desde los primeros años de carrera, pues nos mencionaban que teníamos ejes x, y, z, y con estos podíamos representar figuras, animaciones, videojuegos, en la computadora.

También nos ayuda a entender de una manera muy escueta como es que los programadores de videojuegos o animadores por computadora hacen su trabajo, pues aunque vimos muchos temas, no es ni la mínima parte de lo que en realidad se hace en el mercado, ni es la manera profesional en la que se hacen las cosas, nosotros hacemos un lector de obj, manejamos bibliotecas que nos ayudan y usamos opengl para las representaciones de estos mismos, pero allá afuera se trabaja con software muy sofisticado, que hace lo que nosotros hicimos y de mejor manera, y lo mas importante de todo es que es bajo un estándar de calidad, acá nosotros programamos bajo la tutela del profesor y pudimos entender los fundamentos de lo que hacen los profesionales con herramientas más completas.

Es necesario mencionar que no cualquier persona puede de la nada llegar y programar algo similar, se requieren conocimientos algebraicos, programación, y por experiencia personal tener conocimiento del sistema Linux, pues te va a facilitar mucho el proceso y, sobre todo, una lógica avanzada, pues en cualquier momento te puedes perder en el proceso.

Desarrollo:

El proyecto se llevo a cabo en una máquina virtual ya que en Windows nos dio muchos problemas, se probó con: Ubuntu en un inicio, sin embargo, mi PC no aguantaba, se trababa mucho, luego intentamos en Lubuntu, mejoro, pero no era lo ideal, hasta que encontramos un sistema que no requiriera casi nada de recursos y funcione de manera ideal, todo el proyecto se realizó en Xubuntu.

La base del proyecto es el lector OBJ, este es el que lee los archivos .obj, los cuales son vitales pues son los que manipulamos e imprimimos en pantalla.

Lector Obj

Se divide en 3 clases, las cuales son vertex, face, object.

Diagrama de flujo del lector:

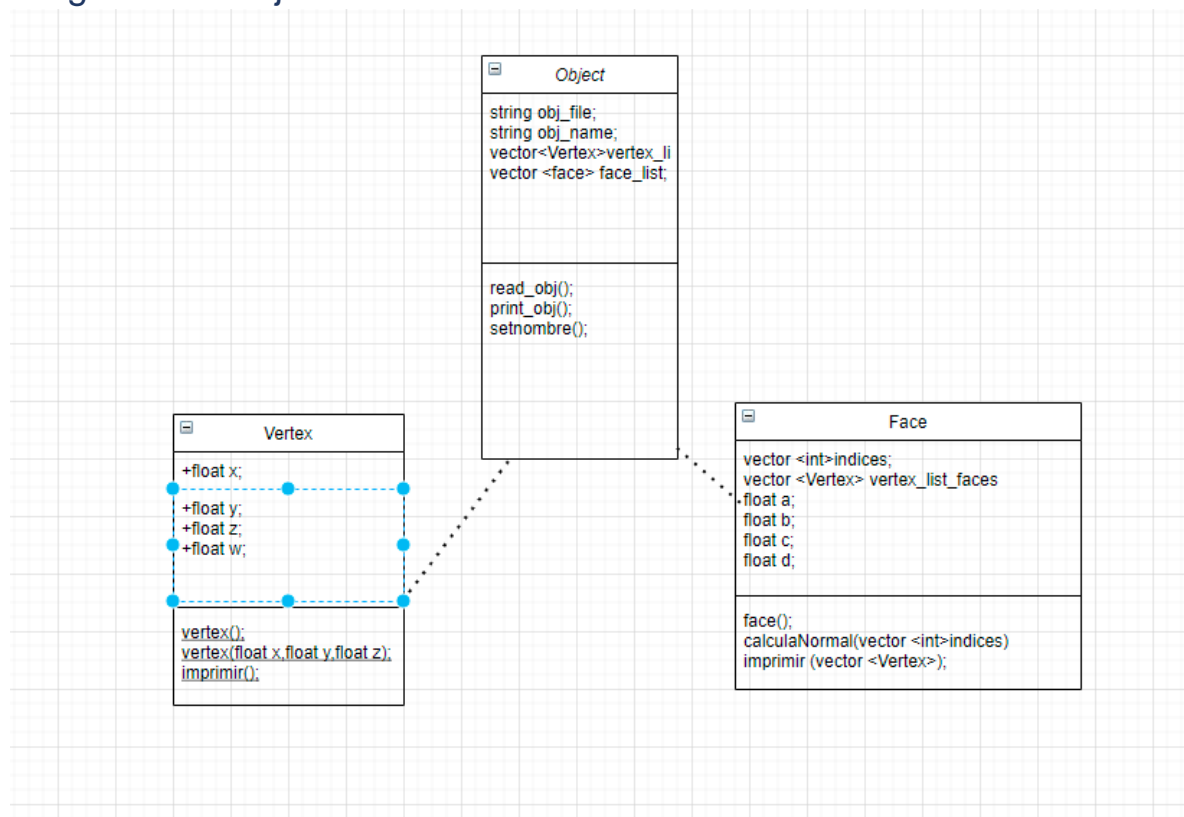
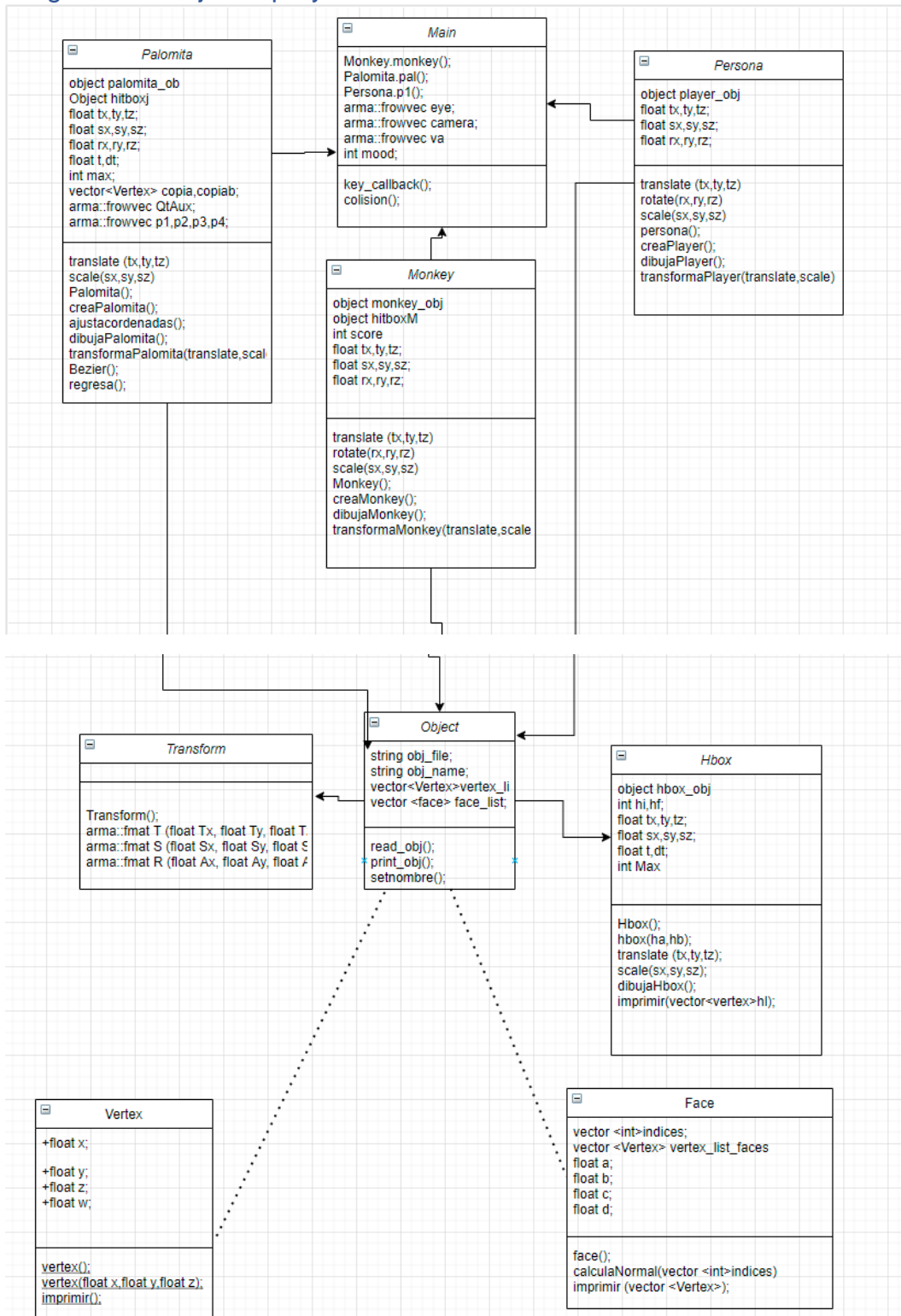


Diagrama de flujo del proyecto:



Object:

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
1 #ifndef OBJECT_HPP
2 #define OBJECT_HPP
3 #include <iostream>
4 #include <fstream>
5 #include <sstream>
6 #include <armadillo>
7 #include <vector>
8 #include "vertex.hpp"
9 #include "face.hpp"
10 using namespace std;
11
12 class Object{
13     public:
14         string obj_file;
15         string obj_name;
16
17         vector <Vertex> vertex_list;
18         vector <Face> face_list;
19
20         Object();
21         void read_obj();
22         void print_obj();
23         void setnombre(const char *nombre);
24 };
25 #endif
```

Se declara una variable `obj_file` guarda el nombre del archivo.

Se declara una variable `obj_name` guarda el nombre de la figura que viene definida dentro del obj.

Se tiene una lista de vértices llamada `vertex_list`.

Se tiene una lista de caras llamada `face_list`.

Un constructor vacío.

La función `setnombre` asigna el nombre del archivo obj a leer.

La función `read_obj` se auxilia de la librería `<sstream>` la cual tiene una función llamada `stringstream`, la cual recibe una secuencia de caracteres y nos permite acceder a cada uno de ellos directamente, básicamente si tenemos:

```
v  0.0  0.0  0.0
```

Nosotros fácilmente podríamos asignar el primer valor: 0.0 a una variable por ejemplo x, el siguiente 0.0 a Y, el ultimo 0.0 a z.

Read_obj extrae línea por línea, y checa si:

En caso de que el primer carácter es “o” significa que lo siguiente que va a leer es el nombre del obj propuesto por el programador que realizo el obj.

En caso de que el primer carácter es “v” significa que lo siguiente que leerá son vértices, los cuales se dividen por espacios, correspondientes a x, y, z.

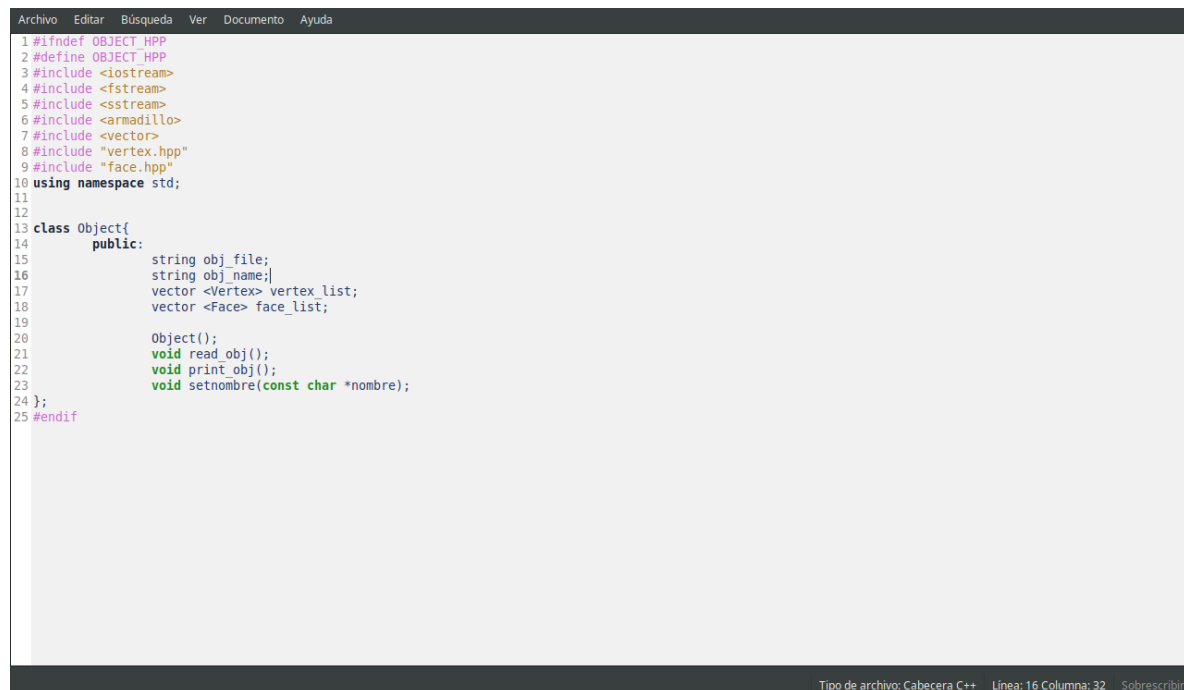
En caso de que el primer carácter es “vn” significa que es la normal de dichos vértices.

En caso de que el primer carácter es “f” significa que tenemos los índices de los vértices que todos juntos, triangulados forman una cara.

Una vez que realiza el proceso de checar que esta leyendo segmenta la línea de código y la guarda cada variable, para después ser manipulada según sea el caso.

Print_obj lo único que realiza es imprimir la información obtenida en consola.

HPP:



```
1 #ifndef OBJECT_HPP
2 #define OBJECT_HPP
3 #include <iostream>
4 #include <fstream>
5 #include <sstream>
6 #include <armadillo>
7 #include <vector>
8 #include "vertex.hpp"
9 #include "face.hpp"
10 using namespace std;
11
12
13 class Object{
14     public:
15         string obj_file;
16         string obj_name;
17         vector <Vertex> vertex_list;
18         vector <Face> face_list;
19
20         Object();
21         void read_obj();
22         void print_obj();
23         void setnombre(const char *nombre);
24 };
25 #endif
```

Tipo de archivo: Cabecera C++ Línea: 16 Columna: 32 [Sobrescribir](#)

C++:

Archivo Editar Búsqueda Ver Documento Ayuda

```
1 #include "object.hpp"
2
3 Object::Object(){
4 }
5
6 void Object::setnombre(const char *file){
7     obj_file=file;
8 }
9
10 void Object::read_obj(){
11     string linea;
12     ifstream archivoOBJ(obj_file);
13
14     while(getline (archivoOBJ, linea)){
15         if(linea.size()>1){
16             if((linea[0] == 'o' || linea[0] == 'g')){
17                 string name =linea.substr(2);
18                 obj_name= name;
19             }
20             if(linea[0] == 'v' && linea[1] != 'n'){
21                 string numeros;
22                 numeros= linea.substr(2);
23                 stringstream num segmentados;
24                 num segmentados<< numeros;
25                 float x,y,z;
26                 num segmentados>> x >> y >> z;
27                 Vertex v= Vertex(x,y,z);
28                 vertex_list.push_back(v);
29             }
30
31             if(linea[0] == 'f'){
32                 Face f;
33                 vector<int> :: iterator it;
34                 int vert;
35                 int prev_vert2= -1;
36                 int prev_vert= -1;
37                 string aux_string;
38                 string nums;
39                 nums=linea.substr(2);
40                 stringstream num_seg;
```

Tipo de archivo: C++ Línea: 42 Columna: 46 Sobrescribir

Archivo Editar Búsqueda Ver Documento Ayuda

```
38         string nums;
39         nums=linea.substr(2);
40         stringstream num_seg;/////////////////
41         //stringstream num_seg_aux;
42         num_seg<<nums;
43         while(!num_seg.eof()){
44             num_seg>> vert;
45             if(prev_vert != vert){
46                 prev_vert = vert;
47                 f.indices.push_back(vert);
48             }
49             num_seg>>aux_string;
50             if(aux_string[0]!='/'){
51                 stringstream num_seg_aux;
52                 num_seg_aux << aux_string;
53                 num_seg_aux >> vert;
54                 if(prev_vert != vert && prev_vert2 != vert){
55                     prev_vert2 = vert;
56                     prev_vert = vert;
57                     f.indices.push_back(vert);
58                 }
59             }
60         }
61         it = f.indices.begin();
62         int prev = *it;
63         it++;
64         for(;it != f.indices.end();++it){
65             int v = *it;
66             prev = v;
67         }
68         face_list.push_back(f);
69     }
70 }
71
72 }
73
74 }
75 archivoOBJ.close();
76 }
77
```

Tipo de archivo: C++ Línea: 66 Columna: 0 Sobrescribir

```
81
82 void Object::print_obj(){
83     string prob = obj_file;
84     prob=prob.substr(7);
85     if(obj_name == "surface"){
86
87         cout << "NOMBRE DE ARCHIVO: " << prob<<endl;
88     }
89     else{
90         cout << "NOMBRE DE ARCHIVO: " << obj_name<<endl;
91     }
92     for(int i=0; i<face_list.size();i++){
93         face_list[i].imprimir(vertex_list);
94         face_list[i].CalculaNormal(vertex_list);
95     }
96 }
```

Vertex:

HPP:

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
1 #ifndef VERTEX_HPP
2 #define VERTEX_HPP
3 #include <iostream>
4 #include <armadillo>
5 using namespace std;
6
7 class Vertex{
8 public:
9     float x;
10    float y;
11    float z;
12    float w;
13    Vertex();
14    Vertex(float xi, float yi, float zi);
15    void imprimir();
16 };
17 #endif

Tipo de archivo: Cabecera C++  Línea: 7 Columna: 13  Sobrescribir
```

Se declaran cuatro variables, x, y, z, w las cuales guardan las respectivas coordenadas, w es una constante = 1, esto para homogeneizar la ecuación.

Existe un constructor vacío.

El constructor con parámetros asignamos los valores de las coordenadas.

Imprimir imprime los vértices x, y, z en consola.

CPP:

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
1 #include "vertex.hpp"
2
3 Vertex::Vertex(){
4     x = y = z = 0.0f;
5     w=1.0;
6 }
7 Vertex::Vertex(float xi, float yi, float zi){
8     x = xi;
9     y = yi;
10    z = zi;
11    w= 1.0;
12 }
13 void Vertex::imprimir(){
14     cout << "x=" << x << "y=" << y << "z=" << z << endl;
15 }
16

Tipo de archivo: C++  Línea: 1 Columna: 0  Sobrescribir
```

Face:

HPP:

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
1 #ifndef FACE_HPP
2 #define FACE_HPP
3 #include <iostream>
4 #include <vector>
5 #include <armadillo>
6 #include "vertex.hpp"
7
8 using namespace std;
9
10 class Face{
11 public:
12
13     vector <int> indices;
14     Face();
15     vector <Vertex> vertex_list_faces;
16     float A,B,C,D;
17     void CalculaNormal(vector <Vertex> vertices);
18     void imprimir(vector<Vertex>);
19 };
20 #endif
Tipo de archivo: Cabecera C++  Línea: 7 Columna: 0  Sobrescribir
```

Tenemos una lista tipo int de índices, pues en esta clase como tal no manejamos datos, se manejan los índices de las caras, son como coordenadas que podemos triangular para que al final la figura impresa tenga un orden y sentido.

Tenemos un constructor vacío.

El método calculaNormal calcula la normal de cada superficie del objeto usando armadillo, biblioteca que nos facilita mucho las cosas, se usa para el producto cruz.

Imprimir es un método que imprime en consola los vértices que se tienen según la cara correspondiente.

CPP:

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
1 #include "face.hpp"
2
3 Face::Face(){
4 }
5
6 void Face::CalculaNormal(vector <Vertex> vertices){
7     Vertex v1 = vertices[(indices[0].vi)-1];
8     Vertex v2 = vertices[(indices[0].vf)-1];
9     Vertex v3 = vertices[(indices[1].vf)-1];
10
11     arma::frowvec v1p = {v1.x, v1.y, v1.z};
12     arma::frowvec v2p = {v2.x, v2.y, v2.z};
13     arma::frowvec v3p = {v3.x, v3.y, v3.z};
14
15     arma::frowvec NF = arma::cross(v2p-v1p, v3p-v1p);
16     A= NF[0];
17     B= NF[1];
18     C= NF[2];
19     D= -((A*v2.x)+(B*v2.y)+(C*v2.z));
20     cout << "Normal F: " << NF << endl;
21 }
22
23 void Face::imprimir(vector <Vertex> vl){
24     cout << "f ";
25     vector<int>:: iterator it_vectorindices = indices.begin();
26     while(it_vectorindices != indices.end()){
27         cout << *it_vectorindices << " ";
28         it_vectorindices++;
29     }
30     cout << endl;
31     for(int i=0; i < indices.size();i++){
32         indices[i].imprimir(vl);
33     }
34 }
35
36
```

Hasta aquí termina el código que implica al lector obj.

Transform:

Esta clase se encarga de realizar transformaciones específicamente escalaciones(s), rotaciones (r), y traslaciones (t) de los objetos involucrados.

HPP:

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
1 #ifndef TRANSFORM_HPP
2 #define TRANSFORM_HPP
3 #include <stdio.h>
4 #include <armadillo>
5
6 class Transform
7 {
8 public:
9     Transform():
10         arma::fmat T(float tx, float ty, float tz);
11         arma::fmat S(float sx, float sy, float sz);
12         arma::fmat R(float ax, float ay, float az, float angle);
13 };
14
15 #endif // FACE_HPP
16
17 |
```

Tipo de archivo: Cabecera C++ Línea: 17 Columna: 0 Sobrescribir

La clase Transform tiene un constructor vacío.

Son tres métodos tipo fmat, en el cual se realizan operaciones matemáticas con ayuda de la librería armadillo, en la cual se realizan operaciones de matrices, estas ayudan al objeto a transformarse según sea el caso y la necesidad.

CPP:

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
1 #include <cmath>
2 #include "Transform.hpp"
3
4 #define PI 3.14159265
5
6 Transform::Transform() {
7
8 }
9
```

```

10 arma::fmat Transform::T(float tx, float ty, float tz) {
11     return( arma::fmat( {{1.0f, 0.0f, 0.0f, tx},
12                          {0.0f, 1.0f, 0.0f, ty},
13                          {0.0f, 0.0f, 1.0f, tz},
14                          {0.0f, 0.0f, 0.0f, 1.0}} )
15 );
16 }
17
18 arma::fmat Transform::S(float sx, float sy, float sz) {
19     return( arma::fmat( {{sx, 0.0f, 0.0f, 0.0f},
20                          {0.0f, sy, 0.0f, 0.0f},
21                          {0.0f, 0.0f, sz, 0.0f},
22                          {0.0f, 0.0f, 0.0f, 1.0}} )
23 );
24 }
25
26 arma::fmat Transform::R(float ax, float ay, float az, float angle) {
27     float ang = angle * PI / 180.0f;
28     arma::fmat Rot;
29
30     Rot.eye(4, 4);
31
32     if (ax)
33         Rot = arma::fmat( {{1.0f, 0.0f, 0.0f, 0.0f},
34                          {0.0f, cosf(ang), -sinf(ang), 0.0f},
35                          {0.0f, sinf(ang), cosf(ang), 0.0f},
36                          {0.0f, 0.0f, 0.0f, 1.0}} );
37
38     if (ay)
39         Rot = arma::fmat( {{cosf(ang), 0.0f, sinf(ang), 0.0f},
40                          {0.0f, 1.0f, 0.0f, 0.0f},
41                          {-sinf(ang), 0.0, cosf(ang), 0.0f},
42                          {0.0f, 0.0f, 0.0f, 1.0}} );
43
44     if (az)
45         Rot = arma::fmat( {{cosf(ang), -sinf(ang), 0.0f, 0.0f},
46                          {sinf(ang), cosf(ang), 0.0f, 0.0f},
47                          {0.0f, 0.0f, 1.0f, 0.0f},
48                          {0.0f, 0.0f, 0.0f, 1.0}} );
49
50     return(Rot);
51 }

```

Hbox:

Esta función es la que auxilia para poder realizar el hitbox correctamente.

HPP:

```
hbox.cpp ▾  hbox.hpp ▾
1 #ifndef HBOX_HPP
2 #define HBOX_HPP
3 #include <iostream>
4 #include "vertex.hpp"
5 using namespace std;
6
7 class Hbox{
8 public:
9     int hi;
10    int hf;
11    float tx,ty,tz;
12    float sx,sy,sz;
13    float t;
14    float dt;
15    int Max;
16    Hbox();
17    //Object hit_box;
18    Hbox(int ha, int hb);
19    void imprimir(vector <Vertex> hl);
20    void translate(float, float, float);
21    void scale(float, float, float);
22    //void dibujaHbox();
23 };
24 #endif
```

Tipo de archivo: Cabecera C++ Línea: 22 Columna: 28 [Sobrescribir](#)

CPP:

```
hbox.cpp ▾  hbox.hpp ▾
1 #include "hbox.hpp"
2
3 Hbox::Hbox(){
4     t=0;
5     dt=0.001;
6     tx=0.0;
7     ty=0.0;
8     tz=0.0;
9     sx=1.0;
10    sy=1.0;
11    sz=1.0;
12    Max=9;
13 }
14
15 Hbox::Hbox(int ha, int hb){
16     hi=ha;
17     hf=hb;
18 }
19 /*
20 void Hbox::dibujaHbox(){
21     vector <int> :: iterator iterador_indices2;
22     vector <Face> :: iterator iterador_caras2;
23
24     glColor3f(0.5, 0.5, 1.0);
25     glBegin(GL_TRIANGLES);
26     for(iterator caras2=hit_box.face_list.begin();iterador_caras2!=hit_box.face_list.end();iterador_caras2++){
27         for(iterator indices2=iterador_caras2->indices.begin();iterador_indices2!= iterador_caras2->indices.end();iterador_indices2++){
28             glVertex3f(hit_box.vertex_list[*iterador_indices2-1].x,hit_box.vertex_list[*iterador_indices2-1].y,hit_box.vertex_list[*iterador_i
29         }
30     }
31     glEnd();
32 }
33 */
34 void Hbox::imprimir(vector <Vertex> hl){
35     cout<< "\t\t hi:"<<hi<<endl; hl[hi-1].imprimir();
36 }
```

Tipo de archivo: C++ Línea: 13 Columna: 1 [Sobrescribir](#)


```
36 void Hbox::imprimir(vector <Vertex> hl){
37
38     cout<< "t\t hi:"<<hi<<endl; hl[hi-1].imprimir();
39     cout<< "t\t hf:"<<hf<<endl; hl[hf-1].imprimir();
40 }
41
42 void Hbox:: scale(float sx, float sy, float sz){
43     this->sx = sx;
44     this->sy = sy;
45     this->sz = sz;
46 }
47
48 void Hbox:: translate(float tx, float ty, float tz){
49     this->tx = tx;
50     this->ty = ty;
51     this->tz = tz;
52 }
53
54
```

Persona:

Esta clase genera lo necesario para representar el objeto persona, este simula tirar la palomita a Monkey.



HPP

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
1 #ifndef PERSONA_HPP
2 #define PERSONA_HPP
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <GL/glu.h>
6 #include <GLFW/glfw3.h>
7 #include <armadillo>
8 #include "vertex.hpp"
9 #include "face.hpp"
10 #include "object.hpp"
11 #include "Transform.hpp"
12 using namespace std;
13
14 class Persona{
15 public:
16     Object player_obj;
17     float tx,ty,tz;
18     float sx,sy,sz;
19     float rx,ry,rz;
20
21     void translate(float tx, float ty, float tz);
22     void rotate(float rx, float ry, float rz);
23     void scale(float sx, float sy, float sz);
24
25     Persona();
26     void creaPlayer();
27     void dibujaPlayer();
28     void transformaPlayer(bool translate, bool scale);
29 };
30 #endif

Tipo de archivo: Cabecera C++  Línea: 1 Columna: 0  Sobrescribir
```

Existe un atributo tipo Object, este guarda la información de “persona.obj”.

Se hacen variables de tipo float las cuales están enfocadas a los tres tipos de transformaciones, primero se declaran variables para translate: tx, ty, tz, para scale: sx, sy, sz, para rotate: rx, ry, rz.

Una vez declaradas las funciones vienen los métodos.

Tenemos un constructor vacío.

El método creaPlayer es donde se inicializa el proceso en donde el obj en cuestión empieza a tomar forma, se le da un nombre, después se lee el objeto con nuestro lector, y se le asignan valores las variables que se usaran en los métodos Translate, rotate, y scale.

DibujaPlayer es el método que se encarga de dibujar el personaje, todo con ayuda de OpenGL, esto con la ayuda de dos ciclos y dos listas, se dibuja con GL_triangles, pues se manejan caras trianguladas.

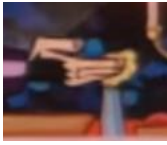
TransformaPlayer realiza las operaciones necesarias para que se ejecute de una manera óptima: Translate, scale, rotate.

CCP:

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
1 #include "persona.hpp"
2
3 Persona:: Persona(){
4 }
5
6 void Persona:: creaPlayer(){
7     player_obj.setnombre("modelos/persona.obj");
8     player_obj.read_obj();
9     tx=0.0;
10    ty=0.0;
11    tz=0.0;
12    rx=0.0;
13    ry=0.0;
14    rz=0.0;
15    sx=1.0;
16    sy=1.0;
17    sz=1.0;
18 }
19
20 void Persona:: dibujaPlayer(){
21
22     vector<int> :: iterator iterador_indices;
23     vector<Face> :: iterator iterador_caras;
24
25     glColor3f(0.5, 0.5, 0.5);
26     glBegin(GL_POLYGON);
27     for(iterador_caras=player_obj.face_list.begin(); iterador_caras!=player_obj.face_list.end(); iterador_caras++){
28         for(iterador_indices=iterador_caras->indices.begin(); iterador_indices!= iterador_caras->indices.end(); iterador_indices++){
29             glVertex3f(player_obj.vertex_list[*iterador_indices-1].x, player_obj.vertex_list[*iterador_indices-1].y, player_obj.vertex_list[*iterador_indices-1].z);
30         }
31     }
32     glEnd();
33 }
34
35 void Persona:: transformaPlayer(bool translate, bool scale){
36
37     Transform Tr= Transform();
38     arma::fmat trans = {{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}};
39
40     if(translate == true){
41         trans = trans * Tr.T(tx, ty, tz);
42     }
43     if(scale == true){
44         trans = trans * Tr.S(sx,sy,sz);
45     }
46
47     for ( unsigned int i=0; i<player_obj.vertex_list.size(); i++ ) {
48         arma::fcolvec v = {{player_obj.vertex_list[i].x},{player_obj.vertex_list[i].y},{player_obj.vertex_list[i].z},{player_obj.vertex_list[i].w}};
49         arma::fcolvec vp = trans * v;
50         player_obj.vertex_list[i]={vp[0]},{vp[1]},{vp[2]}};
51     }
52 }
53
54 void Persona:: rotate(float rx, float ry, float rz){
55     this->rx = rx;
56     this->ry = ry;
57     this->rz = rz;
58 }
59
60 void Persona:: scale(float sx, float sy, float sz){
61     this->sx = sx;
62     this->sy = sy;
63     this->sz = sz;
64 }
65
66 void Persona:: translate(float tx, float ty, float tz){
67     this->tx = tx;
68     this->ty = ty;
69     this->tz = tz;
70 }
```

Palomita

Es el objeto por lanzar, el lanzamiento es una trayectoria curva, en la cual se implemento el concepto de las curvas de Bézier.



HPP:

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
1 #ifndef PALOMITA_HPP
2 #define PALOMITA_HPP
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <time.h>
6 #include <cmath>
7 #include <GL/glu.h>
8 #include <GLFW/glfw3.h>
9 #include <armadillo>
10 #include "vertex.hpp"
11 #include "face.hpp"
12 #include "object.hpp"
13 #include "Transform.hpp"
14 using namespace std;
15
16 class Palomita{
17 public:
18     Object palomita_obj;
19     Object hit_box;
20     arma::frowvec QtAux;
21     arma::frowvec P1,P2,P3,P4;
22     vector <Vertex> copia,cofiab;
23     float tx,ty,tz;
24     float sx,sy,sz;
25     float t;
26     float dt;
27     int Max;
28
29     void translate(float, float, float);
30     void scale(float, float, float);
31
32     Palomita();
33     void creaPalomita();
34     void ajustaCoordenadas();
35     void dibujaPalomita();
36     void transformaPalomita(bool translate, bool scale);
37     void Bezier();
38     void regresa();
39 };
40 #endif

Tipo de archivo: Cabecera C++  Línea:28 Columna:0  Sobrescribir
```

El objeto “palomita.obj” es aquel que se va a mover, por lo tanto lo declaramos un palomita_obj, seguido de eso tenemos un concepto nuevo, pues necesitamos una colisión, en semestres pasados un profesor nos introdujo en el tema de colisiones, y nos mostró un concepto llamado “Hitbox” el cual menciona que es un cubo el cual va a tener relativamente las mismas dimensiones que el personaje que colisionara y que este cubo se va a mover a la par que el objeto que deseamos que colisione, este hitbox lo tiene el objeto Palomita y el objeto Monkey, en el momento que el hitbox de la palomita toque el hitbox de Monkey el objeto Palomita desaparece, y vuelve al inicio.

Para el cálculo de la curva de Bézier se declara QtAux, P1, P2, P3, P4 de tipo frowvec, dos listas de vértices para asegurarnos que si toca el suelo vuelva al origen y si toca al mono también vuelva al origen.

Se hacen variables de tipo float las cuales están enfocadas a los tres tipos de transformaciones, primero se declaran variables para translate: tx, ty, tz, para scale: sx, sy, sz, para rotate: rx, ry, rz.

Las variables t y dt son los incrementos que se usan en Bézier, la variable Max para tener un random fijo.

El método `creaPalomita` es donde se inicializa el proceso en donde el obj en cuestión empieza a tomar forma, se le da un nombre, después se lee el objeto con nuestro lector, y se le asignan valores las variables que se usaran en los métodos `Translate`, `rotate`, y `scale`.

`DibujaPalomita` es el método que se encarga de dibujar el personaje, todo con ayuda de OpenGL, esto con la ayuda de dos ciclos y dos listas, se dibuja con `GL_triangles`, pues se manejan caras trianguladas.

`TransformaPalomita` realiza las operaciones necesarias para que se ejecute de una manera óptima: `Translate`, `scale`, `rotate`.

El método Bézier hace los cálculos de la curva y después de hacer los cambios de coordenada transforma y dibuja la palomita, esto en un ciclo hasta que encuentre el hitbox o el suelo, en caso de que una de estas dos cosas pase manda llamar al método `regresa`, este regresa a la palomita y al hitbox a su posición inicial con ayuda de ajusta coordenadas, este método ayuda a ajustar las coordenadas para que esta operación se haga de manera correcta.

C++:

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
1 #include "palomita.hpp"
2
3 Palomita:: Palomita(){
4
5     t=0;
6     dt=0.001;
7     tx=0.0;
8     ty=0.0;
9     tz=0.0;
10    sx=1.0;
11    sy=1.0;
12    sz=1.0;
13    Max=9;
14 }
15 void Palomita:: ajustaCoordenadas(){
16
17     srand (time(NULL));
18     float x1= (float) ((int)rand()%((Max-1)*(int)pow(10,2)))/pow(10,2)+1;
19     //cout<<x1<<endl;
20
21     P1 = {palomita_obj.vertex_list[(palomita_obj.vertex_list.size())/2].x,palomita_obj.vertex_list[(palomita_obj.vertex_list.size())/2].y,palomita_obj.vertex_list[(palomita_obj.vertex_list.size())/2].z};
22     P2 = {P1[0]+10,P1[1]+5,P1[2]};
23     P4 = {x1,0,0};
24     P3 = {x1,P4[1]+10,0};
25 }
26 void Palomita:: creaPalomita(){
27     palomita_obj.setnombre("modelos/palomita.obj");
28     palomita_obj.read_obj();
29     hit_box.setnombre("modelos/palomitaBox.obj");
30     hit_box.read_obj();
31 }
32
33 void Palomita:: dibujaPalomita(){
34
35     vector<int> :: iterator iterador_indices;
36     vector<Face> :: iterator iterador_caras;
37     //vector<int> :: iterator iterador_indices2;
38     //vector<Face> :: iterator iterador_caras2;
39 }
```

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
39
40     glColor3f(1.0, 1.0, 1.0);
41     glBegin(GL_TRIANGLES);
42     for(iterator_caras=palomita_obj.face_list.begin(); iterator_caras!=palomita_obj.face_list.end(); iterator_caras++){
43         for(iterator_indices=iterator_caras->indices.begin(); iterator_indices!= iterator_caras->indices.end(); iterator_indices++){
44             glVertex3f(palomita_obj.vertex_list[*iterator_indices-1].x, palomita_obj.vertex_list[*iterator_indices-1].y, palomita_obj.vertex_list[*iterator_indices-1].z);
45         }
46     }
47
48     glEnd();
49 }
50
51 void Palomita::Bezier(){
52     arma::fmat MB= {{-1,3,-3,1},{3,-6,3,0},{-3,3,0,0},{1,0,0,0}};
53     arma::fmat GB (4,3);
54     GB.row(0)=P1;
55     GB.row(1)=P2;
56     GB.row(2)=P3;
57     GB.row(3)=P4;
58
59     if(t==0){
60         arma::frowvec T={powf(t,3),powf(t,2),t,1};
61         arma::frowvec Qt = T * MB * GB;
62         QtAux= Qt;
63         //cout<<QtAux<<endl;
64     }
65     if(t<=1){
66         arma::frowvec T={powf(t,3),powf(t,2),t,1};
67         arma::frowvec Qt = T * MB * GB;
68         //cout<<"Qt: " <<Qt<<endl;
69         arma::frowvec D= Qt-QtAux;
70         //cout<<"D: " <<D<<endl;
71         translate(D[0],D[1],D[2]);
72         transformaPalomita(true,false);
73         dibujaPalomita();
74         QtAux=Qt;
75         t= t+dt;
76     }
77     else{
78         if(t>1){
```

Tipo de archivo: C++ Línea: 25 Columna: 1 Sobrescribir

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
76     }
77     else{
78         if(t>1){
79             regresa();
80         }
81     }
82 }
83
84 void Palomita:: regresa(){
85     t=0;
86     palomita_obj.vertex_list=copia;
87     hit_box.vertex_list=copiab;
88     ajustaCoordenadas();
89 }
90
91 void Palomita:: scale(float sx, float sy, float sz){
92     this->sx = sx;
93     this->sy = sy;
94     this->sz = sz;
95 }
96
97 void Palomita:: translate(float tx, float ty, float tz){
98     this->tx = tx;
99     this->ty = ty;
100    this->tz = tz;
101 }
102
103 void Palomita:: transformaPalomita(bool translate, bool scale){
104     Transform Tr= Transform();
105     arma::fmat trans = {{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}};
106
107     if(translate == true){
108         trans = trans * Tr.T(tx, ty, tz);
109     }
110     if(scale == true){
111         trans = trans * Tr.S(sx,sy,sz);
112     }
113
114     for ( unsigned int i=0; i<palomita_obj.vertex_list.size(); i++ ) {
```

Tipo de archivo: C++ Línea: 65 Columna: 17 Sobrescribir

```
113
114     for ( unsigned int i=0; i<palomita_obj.vertex_list.size(); i++ ) {
115         arma::fcolvec v = {{palomita_obj.vertex_list[i].x},{palomita_obj.vertex_list[i].y},{palomita_obj.vertex_list[i].z},{palomita_obj.vertex_list[i].w}};
116         arma::fcolvec vp = trans * v;
117         palomita_obj.vertex_list[i]={vp[0]},{vp[1]},{vp[2]}};
118     }
119     for ( unsigned int i=0; i<hit_box.vertex_list.size(); i++ ) {
120         arma::fcolvec va = {{hit_box.vertex_list[i].x},{hit_box.vertex_list[i].y},{hit_box.vertex_list[i].z},{hit_box.vertex_list[i].w}};
121         arma::fcolvec vpa = trans * va;
122         hit_box.vertex_list[i]={vpa[0]},{vpa[1]},{vpa[2]}};
123     }
124 }
125
```

Tipo de archivo: C++ Línea: 122 Columna: 54 Sobrescribir

Monkey:

Monkey es el objeto que se va a mover con ayuda de las flechas del teclado, y que si el objeto choca con la palomita se la “come”.



HPP:

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
1 #ifndef MONKEY_HPP
2 #define MONKEY_HPP
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <GL/glu.h>
6 #include <GLFW/glfw3.h>
7 #include <armadillo>
8 #include "vertex.hpp"
9 #include "face.hpp"
10 #include "object.hpp"
11 #include "Transform.hpp"
12 using namespace std;
13
14 class Monkey{
15 public:
16     Object monkey_obj;
17     Object hit_boxM;
18     int score;
19     float tx,ty,tz;
20     float sx,sy,sz;
21     float rx,ry,rz;
22
23     void translate(float, float, float);
24     void rotate(float, float, float);
25     void scale(float, float, float);
26
27     Monkey();
28     void creaMonkey();
29     void dibujaMonkey();
30     void transformaMonkey(bool translate, bool scale);
31
32 };
33 #endif

Tipo de archivo: Cabecera C++  Línea: 1 Columna: 0  Sobrescribir
```

El objeto “monkey_obj” es aquel que se va a mover con las teclas, colisionará con la palomita, y esta misma volverá al inicio; Se declaran dos variables tipo Object: monkey_obj y hitboxM.

Se tiene un constructor vacío.

Se hacen variables de tipo float las cuales están enfocadas a los tres tipos de transformaciones, primero se declaran variables para translate: tx, ty, tz, para scale: sx, sy, sz, para rotate: rx, ry, rz.

TransformaMonkey realiza las operaciones necesarias para que se ejecute de una manera óptima: Translate, scale, rotate.

El método creaMonkey es donde se inicializa el proceso en donde el obj en cuestión empieza a tomar forma, se le da un nombre, después se lee el objeto con nuestro lector, y se le asignan valores las variables que se usaran en los métodos Translate, rotate, y scale.

DibujaMonkey es el método que se encarga de dibujar el personaje, todo con ayuda de OpenGL, esto con la ayuda de dos ciclos y dos listas, se dibuja con GL_triangles, pues se manejan caras trianguladas.

C++

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
1 #include "monkey.hpp"
2
3 Monkey:: Monkey(){
4 }
5
6 void Monkey:: creaMonkey(){
7     monkey_obj.setnombre("modelos/monkey.obj");
8     monkey_obj.read_obj();
9     hit_boxM.setnombre("modelos/monkeyBox.obj");
10    hit_boxM.read_obj();
11
12    tx=0.0;
13    ty=0.0;
14    tz=0.0;
15    rx=0.0;
16    ry=0.0;
17    rz=0.0;
18    sx=1.0;
19    sy=1.0;
20    sz=1.0;
21
22 }
23
24 void Monkey:: dibujaMonkey(){
25
26     vector <int> :: iterator iterador_indices;
27     vector <Face> :: iterator iterador_caras;
28
29     glColor3f(0.5, 0.25, 0.0);
30     glBegin(GL_TRIANGLES);
31     for(iterador_caras=monkey_obj.face_list.begin();iterador_caras!=monkey_obj.face_list.end();iterador_caras++){
32         for(iterador_indices=iterador_caras->indices.begin();iterador_indices!= iterador_caras->indices.end();iterador_indices++){
33             glVertex3f(monkey_obj.vertex_list[*iterador_indices-1].x,monkey_obj.vertex_list[*iterador_indices-1].y,monkey_obj.vertex_list[*iterador_indices-1].z);
34         }
35     }
36     glEnd();
37 }
38 }
```



```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
40 void Monkey:: transformaMonkey(bool translate, bool scale){
41     Transform Tr= Transform();
42     arma::fmat trans = {{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}};
43
44     if(translate == true){
45         trans = trans * Tr.T(tx, ty, tz);
46     }
47     if(scale == true){
48         trans = trans * Tr.S(sx,sy,sz);
49     }
50
51     for ( unsigned int i=0; i<monkey_obj.vertex_list.size(); i++ ) {
52         arma::fcolvec v = {{monkey_obj.vertex_list[i].x},{monkey_obj.vertex_list[i].y},{monkey_obj.vertex_list[i].z},{monkey_obj.vertex_list[i].w}};
53         arma::fcolvec vp = trans * v;
54         monkey_obj.vertex_list[i]={vp[0]},{vp[1]},{vp[2]}};
55     }
56     for ( unsigned int i=0; i<hit_boxM.vertex_list.size(); i++ ) {
57         arma::fcolvec va = {{hit_boxM.vertex_list[i].x},{hit_boxM.vertex_list[i].y},{hit_boxM.vertex_list[i].z},{hit_boxM.vertex_list[i].w}};
58         arma::fcolvec vpa = trans * va;
59         hit_boxM.vertex_list[i]={vpa[0]},{vpa[1]},{vpa[2]}};
60     }
61 }
62
63 void Monkey:: rotate(float rx, float ry, float rz){
64     this->rx = rx;
65     this->ry = ry;
66     this->rz = rz;
67 }
68
69 void Monkey:: scale(float sx, float sy, float sz){
70     this->sx = sx;
71     this->sy = sy;
72     this->sz = sz;
73 }
74
75 void Monkey:: translate(float tx, float ty, float tz){
76     this->tx = tx;
77     this->ty = ty;
78     this->tz = tz;
79 }
```

Tipo de archivo: C++ Línea: 28 Columna: 0 Sobrescribir

Main:

En el main tenemos dos funciones principales, key_callback, colisión, y lo necesario para dibujar los objetos.

Key_callback:

```
107 void key_callback(GLFWwindow* window, int ket, int scancode, int action, int mods){
108     if(mood ==0 || mood==1){
109         if(ket== GLFW_KEY_UP && action == GLFW_PRESS){
110             monkey.translate(0.0,0.8,0.0);
111             monkey.transformaMonkey(true,false);
112         }
113         if(ket== GLFW_KEY_RIGHT && action == GLFW_PRESS){
114             monkey.translate(0.8,0.0,0.0);
115             monkey.transformaMonkey(true,false);
116         }
117
118         if(ket== GLFW_KEY_LEFT && action == GLFW_PRESS){
119             monkey.translate(-0.8,0.0,0.0);
120             monkey.transformaMonkey(true,false);//////////
121         }
122         if(ket== GLFW_KEY_DOWN && action == GLFW_PRESS){
123             monkey.translate(0.0,-0.8,0.0);
124             monkey.transformaMonkey(true,false);
125         }
126     }
127     if(mood==2){
128         if(ket== GLFW_KEY_UP && action == GLFW_PRESS){
129             monkey.translate(-0.8,0.0,0.0);
130             monkey.transformaMonkey(true,false);
131         }
132         if(ket== GLFW_KEY_RIGHT && action == GLFW_PRESS){
133             monkey.translate(0.0,-0.8,0.0);
134             monkey.transformaMonkey(true,false);
135         }
136
137         if(ket== GLFW_KEY_LEFT && action == GLFW_PRESS){
138             monkey.translate(0.0,0.8,0.0);
139             monkey.transformaMonkey(true,false);
140         }
141         if(ket== GLFW_KEY_DOWN && action == GLFW_PRESS){
142             monkey.translate(0.8,0.0,0.0);
143             monkey.transformaMonkey(true,false);
144     }
```

Detecta las teclas propuestas (las flechas), una vez presionada la tecla deseada el programa detecta si es hacia arriba, abajo, izquierda o derecha y se hace la traslación.

Así mismo hay 3 modos de cámara, con la tecla 0 es la vista normal, con la tecla 1 es una vista inversa y con la tecla 2 es la vista desde la parte de arriba, todo esto gracias a gluLookAt.

```

146         if(ket== GLFW_KEY_0 && action == GLFW_PRESS){
147             cout << "vista normal " << endl;
148             eye = {0.0, 0.0, 10.0};
149             camera = {0.0, 0.0, 0.0};
150             va = {0.0, 1.0, 0.0};
151             mood =0;
152         }
153         if(ket== GLFW_KEY_1 && action == GLFW_PRESS){
154             cout << "vista inversa " << endl;
155             eye = {0.0, 10.0, 0.0};
156             camera = {0.0, 0.0, 10.0};
157             va = {0.0, 0.0, -1.0};
158             pl.translate(7.6,0,0);
159             mood=1;
160         }
161
162         if(ket== GLFW_KEY_2 && action == GLFW_PRESS){
163             cout << "vista arriba " << endl;
164             eye = {0.0, 10.0, 0.0};
165             camera = {0.0, 0.0, 0.0};
166             va = {-1.0, 0.0, 0.0};
167             mood=2;
168         }
169     }

```

Colisión:

Detecta si el hitbox de la palomita es interceptado por el hitbox de Monkey.

```

172 void colision(){
173     //Ecuación del plano Ax+By+Cz-D=0 Normal (A,B,C)
174     bool bandera=false;
175     int i=0;
176     while(bandera==false && i<monkey.hit_boxM.vertex_list.size()){
177         Vertex punto= monkey.hit_boxM.vertex_list[i];
178         bandera=true;
179         int j=0;
180         while(bandera==true && j<pal.hit_box.face_list.size()){
181             Face cara= pal.hit_box.face_list[j];
182             float valor= (cara.A*punto.x)+(cara.B*punto.y)+(cara.C*punto.z)-cara.D;
183             if(valor>0){
184                 //cout << "no choque"<< endl;
185                 bandera=false;
186             }
187             j++;
188         }
189         i++;
190     }
191     if(bandera== false){
192         //cout <<" si hubo choque " << endl;
193         pal.regresa();
194     }
195 }

```

Se evalúa la ecuación del plano para que cualquier punto que pertenezca a la hitbox de Monkey sea evaluado en la ecuación del plano con valores en la caja de la palomita y saber si cuando es menor a 0 ó 0 se encuentran dentro de ese plano y por lo tanto hay colisión. Todo dentro de un while con banderas porque si un punto no está chocando con la cara no existe dicha colisión.

CPP

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
1
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <GL/glu.h>
5 #include <GLFW/glfw3.h>
6 #include <armadillo>
7 #include "include/persona.hpp"
8 #include "include/monkey.hpp"
9 #include "include/palomita.hpp"
10 #include "include/hbox.hpp"
11 #include "include/face.hpp"
12 #include "include/object.hpp"
13 #include "include/Transform.hpp"
14
15 void key_callback(GLFWwindow* window,int ket,int scancode,int action,int mods);
16 void colision();
17 Monkey monkey = Monkey();
18 Palomita pal= Palomita();
19 Persona pl= Persona();
20 arma::frowvec eye = {0.0, 0.0, 10.0};
21 arma::frowvec camera = {0.0, 0.0, 0.0};
22 arma::frowvec va = {0.0, 1.0, 0.0};
23 int mood=0;
24
25 int main( void ){
26     GLFWwindow* window;
27
28     if( !glfwInit() ){
29         fprintf( stderr, "Fallo al inicializar GLFW\n" );
30         getchar();
31         return -1;
32     }
33
34     window = glfwCreateWindow(800, 500, "Atrapala bro", NULL, NULL);
35     if( window == NULL ) {
36         fprintf( stderr, "Fallo al abrir la ventana de GLFW.\n" );
37         getchar();
38         glfwTerminate();
39         return -1;
40     }
41 }
```

Tipo de archivo: C++ Línea: 1 Columna: 0 Sobrescribir

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
42     glfwMakeContextCurrent(window);
43     glfwSetInputMode(window, GLFW_STICKY_KEYS, GL_TRUE);
44     glEnable(GL_DEPTH_TEST);
45     glDepthFunc(GL_LESS);
46     glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
47
48 // Proyecciones
49     glMatrixMode(GL_PROJECTION);
50     glLoadIdentity();
51
52     int width, height;
53     glfwGetFramebufferSize(window, &width, &height);
54
55
56 // Proyección en paralelo
57     glViewport(0, 0, width, height);
58     glOrtho(-10, 10, 0.0, 10.0, -10.0, 10.0);
59
60     glMatrixMode(GL_MODELVIEW);
61     glLoadIdentity();
62
63     pl.creaPlayer();
64     pl.scale(0.9,0.9,0.9);
65     pl.translate(-7.6,0,0);
66     pl.rotate(0,0,0);
67     pl.transformaPlayer(true,true);
68
69     monkey.creaMonkey();
70     monkey.scale(2.5,2.5,2.5);
71     monkey.translate(1.0,0,0);
72     monkey.rotate(0,0,0);
73     monkey.transformaMonkey(true,true);
74
75     pal.creaPalomita();
76     pal.translate(-7.05,2.68,0);
77     pal.transformaPalomita(true,false);
78     pal.ajustaCoordenadas();
79     pal.copia= pal.palomita.obj.vertex_list;
80     pal.copiab=pal.hit_box.vertex_list;
81 }
```

Tipo de archivo: C++ Línea: 74 Columna: 8 Sobrescribir

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
81
82     do {
83         glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
84         glMatrixMode(GL_MODELVIEW);
85         glfwSetKeyCallback(window,key_callback);
86         glLoadIdentity();
87         gluLookAt(eye[0], eye[1], eye[2], camera[0], camera[1], camera[2], va[0],va[1] ,va[2]);
88
89         monkey.dibujaMonkey();
90         pl.dibujaPlayer();
91
92         pal.Bezier();
93         colision();
94
95         glfwPollEvents();
96         glfwFlush();
97         glfwSwapBuffers(window);
98
99
100     } while( glfwGetKey(window, GLFW_KEY_ESCAPE ) != GLFW_PRESS && glfwWindowShouldClose(window) == 0 );
101
102     glfwTerminate();
103
104     return 0;
105 }
106
107 void key_callback(GLFWwindow* window,int ket,int scancode,int action,int mods){
108     if(mood ==0 || mood==1){
109         if(ket== GLFW_KEY_UP && action == GLFW_PRESS){
110             monkey.translate(0.0,0.8,0.0);
111             monkey.transformaMonkey(true,false);
112         }
113         if(ket== GLFW_KEY_RIGHT && action == GLFW_PRESS){
114             monkey.translate(0.8,0.0,0.0);
115             monkey.transformaMonkey(true,false);
116         }
117
118         if(ket== GLFW_KEY_LEFT && action == GLFW_PRESS){
119             monkey.translate(-0.8,0.0,0.0);
120             monkey.transformaMonkey(true,false);
121         }
122     }
123 }
```

Tipo de archivo: C++ Línea: 64 Columna: 30 [Sobrescribir](#)

```
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
121     }
122     if(ket== GLFW_KEY_DOWN && action == GLFW_PRESS){
123         monkey.translate(0.0,-0.8,0.0);
124         monkey.transformaMonkey(true,false);
125     }
126 }
127 if(mood==2){
128     if(ket== GLFW_KEY_UP && action == GLFW_PRESS){
129         monkey.translate(-0.8,0.0,0.0);
130         monkey.transformaMonkey(true,false);
131     }
132     if(ket== GLFW_KEY_RIGHT && action == GLFW_PRESS){
133         monkey.translate(0.0,-0.8,0.0);
134         monkey.transformaMonkey(true,false);
135     }
136
137     if(ket== GLFW_KEY_LEFT && action == GLFW_PRESS){
138         monkey.translate(0.0,0.8,0.0);
139         monkey.transformaMonkey(true,false);
140     }
141     if(ket== GLFW_KEY_DOWN && action == GLFW_PRESS){
142         monkey.translate(0.8,0.0,0.0);
143         monkey.transformaMonkey(true,false);
144     }
145 }
146 if(ket== GLFW_KEY_0 && action == GLFW_PRESS){
147     cout << "Vista normal " << endl;
148     eye = {0.0, 0.0, 10.0};
149     camera = {0.0, 0.0, 0.0};
150     va = {0.0, 1.0, 0.0};
151     mood =0;
152 }
153 if(ket== GLFW_KEY_1 && action == GLFW_PRESS){
154     cout << "Vista inversa " << endl;
155     eye = {0.0, 10.0, 0.0};
156     camera = {0.0, 0.0, 10.0};
157     va = {0.0, 0.0, -1.0};
158     pl.translate(7.6,0,0);
159     mood=1;
160 }
161 }
```

Tipo de archivo: C++ Línea: 120 Columna: 77 [Sobrescribir](#)

```

Archivo  Editar  Búsqueda  Ver  Documento  Ayuda
156         camera = {0.0, 0.0, 10.0};
157         va = {0.0, 0.0, -1.0};
158         p1.translate(7.6,0,0);
159         mood=1;
160     }
161
162     if(key== GLFW_KEY_2 && action == GLFW_PRESS){
163         cout << "Vista arriba " << endl;
164         eye = {0.0, 10.0, 0.0};
165         camera = {0.0, 0.0, 0.0};
166         va = {-1.0, 0.0, 0.0};
167         mood=2;
168     }
169 }
170
171 void colision(){
172     //Ecuación del plano Ax+By+Cz-D=0 Normal (A,B,C)
173     bool bandera=false;
174     int i=0;
175     while(bandera==false && i<monkey.hit_boxM.vertex_list.size()){
176         Vertex punto= monkey.hit_boxM.vertex_list[i];
177         bandera=true;
178         int j=0;
179         while(bandera==true && j<pal.hit_box.face_list.size()){
180             Face cara= pal.hit_box.face_list[j];
181             float valor= (cara.A*punto.X)+(cara.B*punto.y)+(cara.C*punto.z)-cara.D;
182             if(valor>0){
183                 //cout << "no choque"<< endl;
184                 bandera=false;
185             }
186             j++;
187         }
188         i++;
189     }
190     if(bandera== false){
191         //cout << " si hubo choque " << endl;
192         pal.regresa();
193     }
194 }
195

```

OpenGL

OpenGL (Open Graphics Library) es una API multiplataforma de gráficos que especifica una interfaz de software estándar para hardware de procesamiento de gráficos 3D.

GLFW es una librería open source, multiplataforma para OpenGL que provee una API para crear ventanas, contextos, superficies y recibir entradas de eventos. Soporta lenguaje C.

Funciones utilizadas:

- `GLFWwindow* window = glfwCreateWindow (640, 480, "My Title", NULL, NULL);`
Crear la ventana del proyecto.
- `glOrtho:`
Crea una multiplicación de matrices para producir una proyección en paralelo.
- `glLoadIdentity:`
Reemplazar la matriz actual por la matriz identidad.
- `glMatrixMode:`
Especificar cual matriz es la actual.
- `glClear:`
Limpia el buffer para presentar los nuevos datos.
- `gluLookAt:`
Define la vista de transformación.
- `glfwPollEvents:`
Reproduce aquellos eventos que se encuentran en la cola de eventos.
- `glFlush:`
Forza la ejecución de comandos GL en un tiempo finito.
- `glfwSwapBuffers:`
Intercambia el buffer delantero y trasero a la ventana.
- `glColor:`
Asignar el color.

- `glColor3f`:
Color en flotantes.
- `glBegin`:
Delimita los vértices de una primitiva.
- `glVertex`:
Especifica un vértice.
- `glfwGetKey`:
Lanza el estado de la última tecla presionada.
- `glfwTerminate`:
Destruya las ventanas activas.

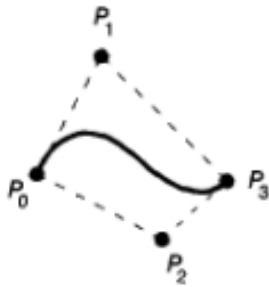
Desarrollo específico (funciones)

Tiro parabólico: Curvas de Bézier

Estas curvas tienen 4 puntos, p_0 , p_1 , p_2 y p_3 .

P_0 y p_3 son punto inicial y punto final respectivamente.

P_1 y p_2 son puntos de control.



```
50 void Palomita::Bezier(){
51     arma::fmat MB= {{-1,3,-3,1},{3,-6,3,0},{-3,3,0,0},{1,0,0,0}};
52     arma:: fmat GB (4,3);
53     GB.row(0)=P1;
54     GB.row(1)=P2;
55     GB.row(2)=P3;
56     GB.row(3)=P4;
57
58     if(t==0){
59         arma:: frowvec T={powf(t,3),powf(t,2),t,1};
60         arma:: frowvec Qt = T * MB *GB;
61         QtAux= Qt;
62         //cout<<QtAux<<endl;
63     }
64     if(t<=1){
65         arma:: frowvec T={powf(t,3),powf(t,2),t,1};
66         arma:: frowvec Qt = T * MB *GB;
67         //cout<<"Qt: "<<Qt<<endl;
68         arma:: frowvec D= Qt-QtAux;
69         //cout<<"D: "<<D<<endl;
70         translate(D[0],D[1],D[2]);
71         transformaPalomita(true,false);
72         dibujaPalomita();
73         QtAux=Qt;
74         t= t+dt;
75     }
76     else{
77         if(t>1){
78             regresa();
79         }
80     }
81 }
```

Prueba de ello es el método en la clase palomita con el nombre Bézier, se declara la matriz de Bézier para la multiplicación y el calculo de los puntos por los cuales pasara la palomita.

Delta t es la velocidad a la cual el objeto se trasladará de punto "a" a punto "b".

Esta es la matriz de Bézier:

Carrying out the multiplication $M_B = M_H \cdot M_{HB}$ gives

$$M_B = M_H \cdot M_{HB} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix},$$

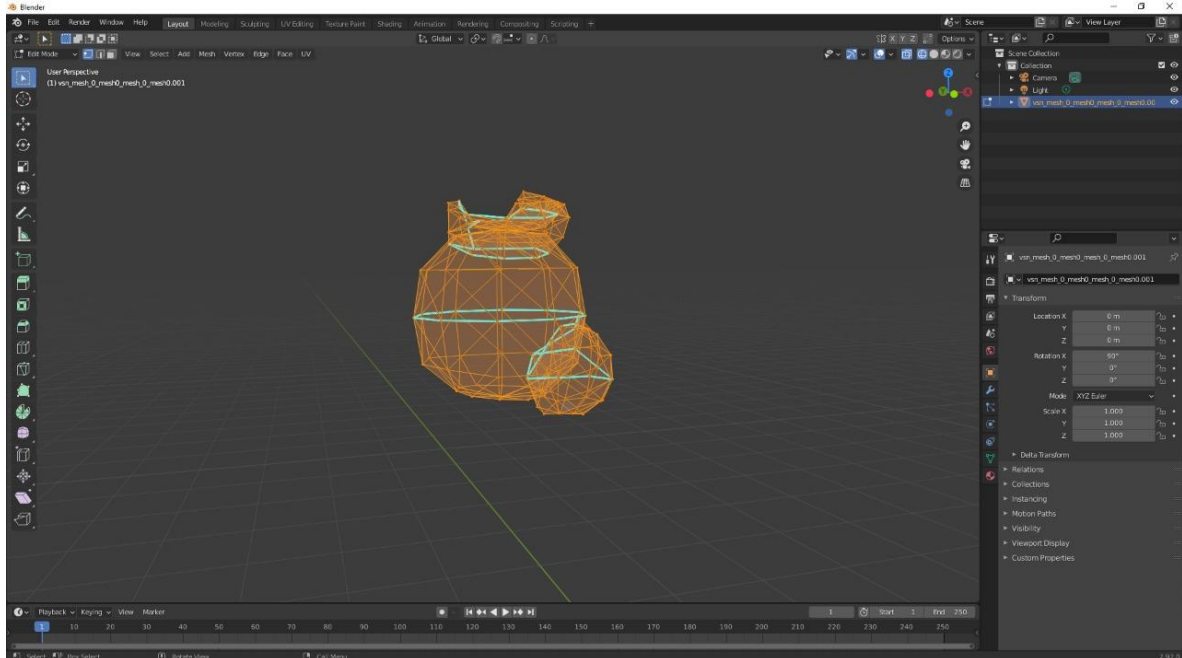
and the product $Q(t) = T \cdot M_B \cdot G_B$ is

$$Q(t) = (1 - t)^3 P_1 + 3t(1 - t)^2 P_2 + 3t^2(1 - t) P_3 + t^3 P_4.$$

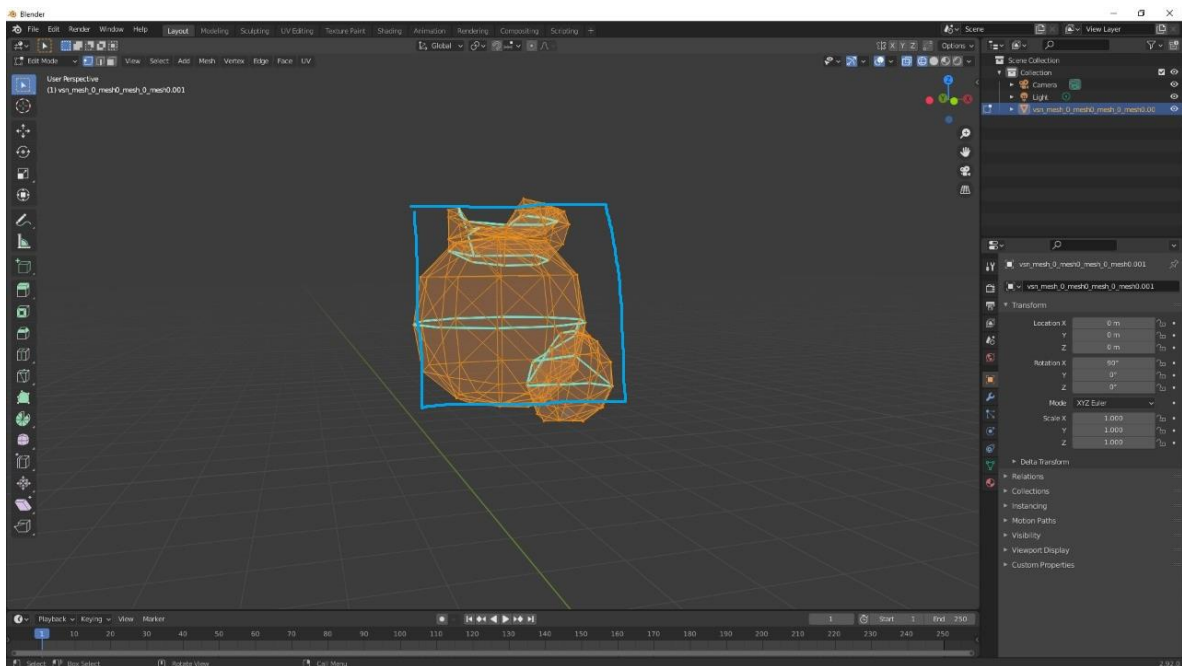
Colisión:

Lo que hacemos es detectar cuando el objeto “a” se intercepta con el objeto “b” con ayuda de un cubo que encasilla los objetos en cuestión y cuando el cubo a en cierto punto tiene la misma coordenada que el cubo b se dice que hay una colisión.

Por ejemplo, acá está el objeto es el objeto lanzado:



Sin embargo, lo que hacemos es “encasillarlo” en un cubo como se representa a continuación:



Esto se hace de igual manera con el “Monkey.obj”, lo que sucede es:

Se evalúa la ecuación del plano para que cualquier punto que pertenezca a la hitbox “a” sea evaluado en la ecuación del plano con valores en la hitbox “b” y saber si cuando es menor a 0 ó 0 se encuentran dentro de ese plano y por lo tanto hay colisión.



Problemas:

- El problema mas grande que enfrente al hacer este proyecto fue usar Windows en un inicio, la verdad me resistía, pero en el momento que se pidió armadillo y opengl en la maquina no pude continuar más, allí me decidí hacer el proyecto en Linux, y todo fluyo.
- El segundo problema mas grande al que me enfrente fue mi lector obj, pues no tenía pies ni cabeza, llegue al punto en el que yo pensaba que funcionaba, pero a la hora de almacenar los datos en realidad no lo estaba haciendo, estaba simplemente imprimiendo un archivo y ya, tuve que arreglar mi lector obj y solo así pude continuar de manera exitosa.
- El tercer problema al que me enfrente fue implementar lo visto en clase, en especial las curvas de Bézier, pues no tenía mucha idea de cómo hacerlo.
- El cuarto problema no fue tan grave, pues se resolvió relativamente rápido, esto fue como hacer que con las teclas mi objeto se moviera, e investigando encontré `key_callback` y se pudo solucionar fácil.
- Sin embargo, el quinto problema era el más importante, la colisión sin embargo preguntando a mis compañeros de otras clases me comentaron que el profesor Ignacio tenía un video al respecto y combinado de otros videos pude implementarlo de buena manera.
- El programa arrojaba `segmentation fault`: Se solucionó borrando en repetidas ocasiones el `makefile` (Esto fue muy recurrente).
- Los `.obj` tuve que modificarlos varias veces, pues los descargue y no los modifique antes de ingresarlos al lector.
- En muchas ocasiones los obj no se mostraban, allí me di cuenta de que el problema ya no eran los obj, si no era el lector. Se soluciono con la función `Split`, esta viene en la biblioteca `sstream`.
- Coloque mal en su momento los `hpp`.
- Antes de los `hpp` no ponía `include/`, por eso no podía ingresar a los archivos de la carpeta `include`.
- No tenía un `makefile` perfecto, después lo tuve.

Conclusiones:

Fue un proyecto que en mi experiencia personal fue muy estresante, desde el principio tuve complicaciones, pero tuve que sobreponerme a los problemas, si algo puedo rescatar de esta materia es: Para programar usa Linux, para todo lo demás Windows.

Asimismo, pude comprender una fracción de conocimiento de los que en verdad se dedican a esto, todo lo que tienen que comprender para que las animaciones o los videojuegos tengan cierta lógica, pues de lo que rescato es que se intenta simular situaciones que pasan en la vida real, los movimientos, las caídas, la luz, la reflexión, etc.

De manera personal desde la preparatoria no le veía cierta utilidad al cálculo que nos enseñaban, a los problemas con matrices, etc. Sin embargo, esta materia me hizo ponerme a prueba y cambiar de idea y de concepción, sobre las matemáticas aplicadas, pues, aunque sea difícil de creer incluso en la carrera seguía sin saber por que tengo que saber de operaciones de matrices y de cálculo.

A simple vista el proyecto parece simple, sin embargo, a sido de los proyectos más difíciles a los que me e enfrentado, la lógica no es nada sencilla para este proyecto, los cálculos, las proyecciones, las conexiones, las representaciones, en fin, este proyecto en lo personal es de un nivel de dificultad muy alto.

Referencias:

- S/A, "GLFW". <https://www.glfw.org/docs/latest/index.html>, 2021.
- S/A- "Arma SourceForge". <http://arma.sourceforge.net/>, S/A.
- Joey de Vries, "LearnOpenGL". <https://learnopengl.com/In-Practice/2D-Game/Collisions/Collision-detection>, 2015
- José Ignacio Nuñez Varela, "Youtube". <https://www.youtube.com/watch?v=BJ-6aIEKrG4>, 2021.
- Alfredo Benavides, "Glosario". <https://glosario.mott.pe/disenio/palabras/curva-bezier>, S/A.
- S/A. "Cplusplus". http://www.cplusplus.com/reference/new/bad_alloc/, 2020.