Universidad Nacional Autónoma de México

Facultad de Ingeniería

Computer Graphics and

Human Computer Interaction Laboratory

Technical Manual

Professor: Eng. Carlos Aldair Roman Balbuena

Student: Chávez Sánchez Juan Daniel

ID: 316350866

Laboratory ID: 9

2022-2 semester

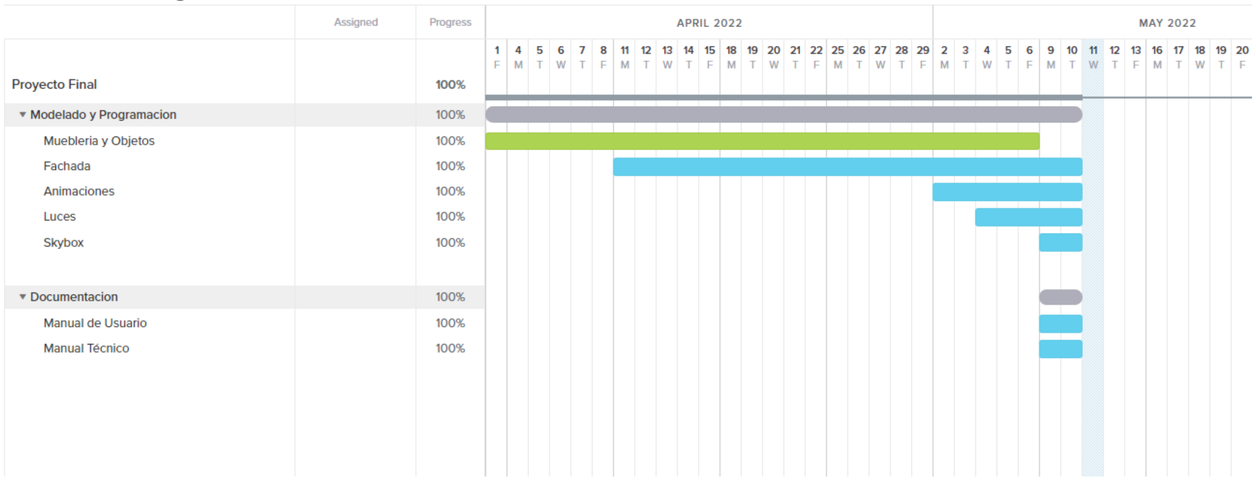
May 11th, 2022, Ciudad Universitaria, CDMX.

Content

Objective

The following document will try to explain the way of working, the tools used, the programming and calculations that were used throughout the realization of the final project in order that the user can learn more about the software.

Gantt Diagram



	Assigned	Progress
Proyecto Final		100%
▾ Modelado y Programacion		100%
Muebleria y Objetos		100%
Fachada		100%
Animaciones		100%
Luces		100%
Skybox		100%
▾ Documentacion		100%
Manual de Usuario		100%
Manual Técnico		100%

Project Scopes

The final project of the subject "Computer Graphics and Human Computer Interaction" has as final public the professor of the same, Eng. Carlos Aldair Roman Balbuena as well as all those who have access to the software made with the purpose of checking only the knowledge achieved in the course thus limiting itself to the purely conceptual native use of OpenGL, modeling, texturing and export of the models made in OBJ format as well as its import, display and animation using Visual Studio 2022.

Limitations

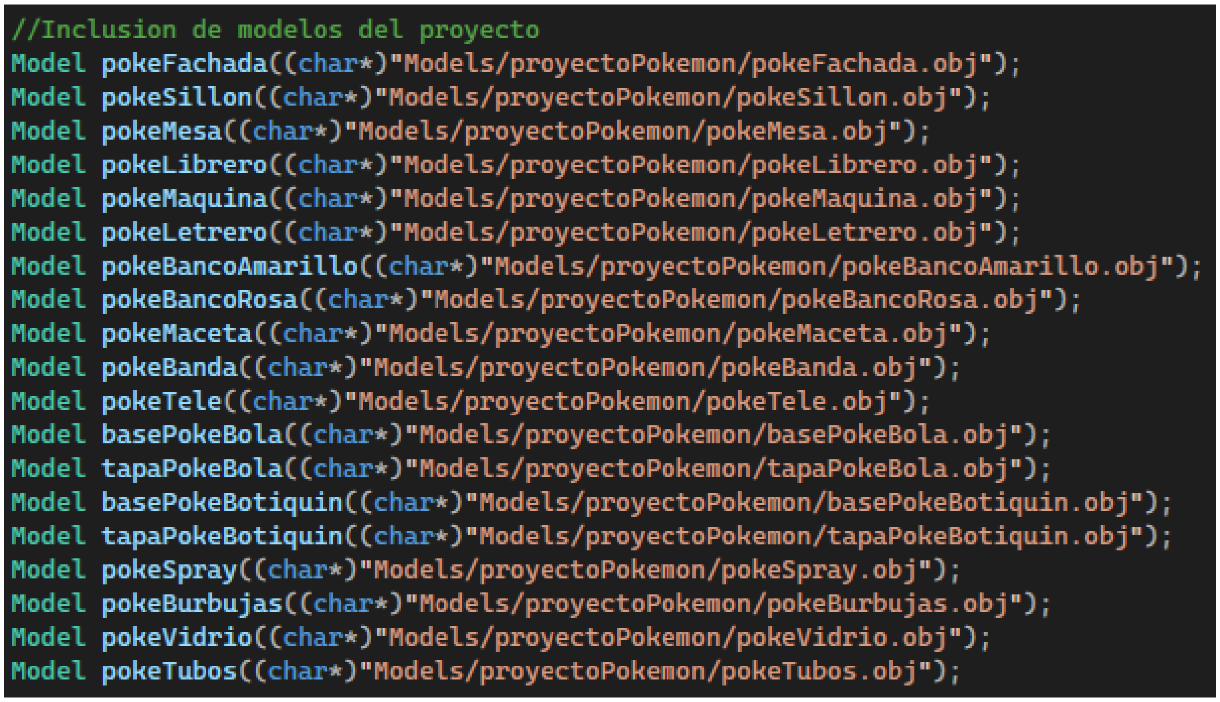
The final project presented can only be built on Windows operating systems and with the minimum requirements set out in the user manual. Likewise, within the project, the actions allowed by using the keys set there to view the animations are the only actions allowed outside the first-person camera control.

Code documentation

For the documentation of the code it has to be taken into account that for this project it was decided to represent a pokemon center of the game in its Omega Ruby and Alpha Saphire versions, so all the models corresponding to the furniture and the façade should have complied with this characteristic. In addition, each and every one of the animations must correspond to an animation seen in the same game.

The software contains 5 animations, 3 simple ones corresponding to the modeling of the pokeballs, the television and the LED sign that is shown rotating in the delivery view for Omega Ruby, and two complex ones corresponding to the bubbles that come out of the tubes at the sides of the structure and to the opening of a first aid kit that takes out a first aid spray and puts it on the table.

To begin then, it will be shown the way in which the models were imported inside the code which are with their respective textures inside the folder "Models" that is inside the same Visual Studio project:



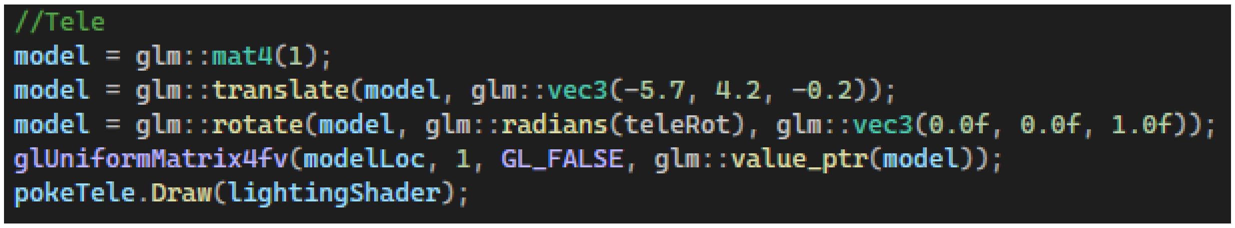
```
//Inclusion de modelos del proyecto
Model pokeFachada((char*)"Models/proyectoPokemon/pokeFachada.obj");
Model pokeSillon((char*)"Models/proyectoPokemon/pokeSillon.obj");
Model pokeMesa((char*)"Models/proyectoPokemon/pokeMesa.obj");
Model pokeLibrero((char*)"Models/proyectoPokemon/pokeLibrero.obj");
Model pokeMaquina((char*)"Models/proyectoPokemon/pokeMaquina.obj");
Model pokeLetrero((char*)"Models/proyectoPokemon/pokeLetrero.obj");
Model pokeBancoAmarillo((char*)"Models/proyectoPokemon/pokeBancoAmarillo.obj");
Model pokeBancoRosa((char*)"Models/proyectoPokemon/pokeBancoRosa.obj");
Model pokeMaceta((char*)"Models/proyectoPokemon/pokeMaceta.obj");
Model pokeBanda((char*)"Models/proyectoPokemon/pokeBanda.obj");
Model pokeTele((char*)"Models/proyectoPokemon/pokeTele.obj");
Model basePokeBola((char*)"Models/proyectoPokemon/basePokeBola.obj");
Model tapaPokeBola((char*)"Models/proyectoPokemon/tapaPokeBola.obj");
Model basePokeBotiquin((char*)"Models/proyectoPokemon/basePokeBotiquin.obj");
Model tapaPokeBotiquin((char*)"Models/proyectoPokemon/tapaPokeBotiquin.obj");
Model pokeSpray((char*)"Models/proyectoPokemon/pokeSpray.obj");
Model pokeBurbujas((char*)"Models/proyectoPokemon/pokeBurbujas.obj");
Model pokeVidrio((char*)"Models/proyectoPokemon/pokeVidrio.obj");
Model pokeTubos((char*)"Models/proyectoPokemon/pokeTubos.obj");
```

Figure 1. Importing models into Visual Studio.

It is under his path within the project how Visual Studio 2022 finds the models and manages to import the OBJ into the job. Once this is done, each of the models are drawn within our scene that will represent the job. On this occasion this document will show those objects that have an animation since the import into the other models and their drawing is exactly the same as these last ones. It is under your direction within the project how Visual Studio 2022 finds the models and manages to import the OBJ within the job. Once this is done, each of the models are drawn inside our scene that will represent the job. On this occasion this document will show those objects that have an animation since the import into the other models and their drawing is exactly the same as the latter.

Simple Animations

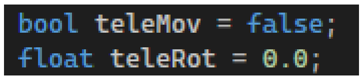
TV Animation



```
//Tele
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-5.7, 4.2, -0.2));
model = glm::rotate(model, glm::radians(teleRot), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
pokeTele.Draw(lightingShader);
```

Figure 2. Drawing of models within the project and imposition of behaviors.

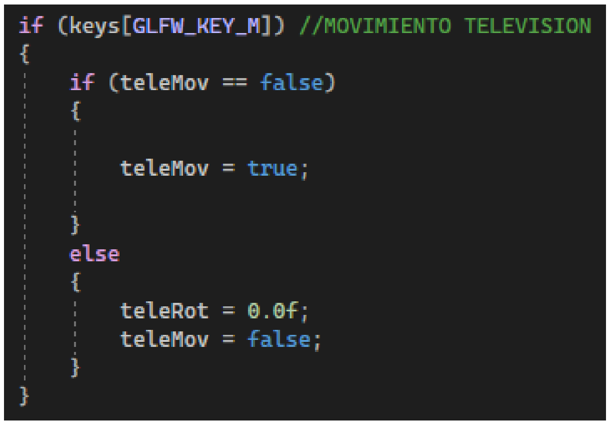
In general, all objects with the exception of those repeated and that are part of the environment, a reset is made to the matrix of positions to adjust it to the position (0,0,0) that is to say, in the center of the scene to be able to make correct use of its pivot once they are drawn within our work. It is with the translation that we establish the position we want it to have within our space and with the rotation the animation we want it to do. In this occasion the television is hanging from the ceiling with the help of two handles which allow it to slide horizontally; to achieve this we required a variable to represent the desired rotation for the television and a second Boolean variable to know when the TV is moving.



```
bool teleMov = false;
float teleRot = 0.0;
```

Figure 2.1. Variables created for animation.

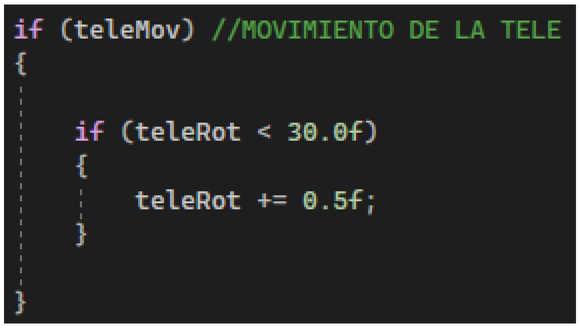
The conditionals establish the start and end points of the animation and also the restart of the animation in its original position, in this case, the M key is used to do this:



```
if (keys[GLFW_KEY_M]) //MOVIMIENTO TELEVISION
{
    if (teleMov == false)
    {

        teleMov = true;

    }
    else
    {
        teleRot = 0.0f;
        teleMov = false;
    }
}
```

Figure 2.2. TV movement conditionals.

The animation is based on the rotation of the Z axis and that it is not greater than 30 degrees which will increase by 0.5 in 0.5 until it reaches 30 degrees and then the degrees are reset to zero.



```
if (teleMov) //MOVIMIENTO DE LA TELE
{

    if (teleRot < 30.0f)
    {
        teleRot += 0.5f;
    }

}
```

Figure 2.3. Steady increase in television rotation.

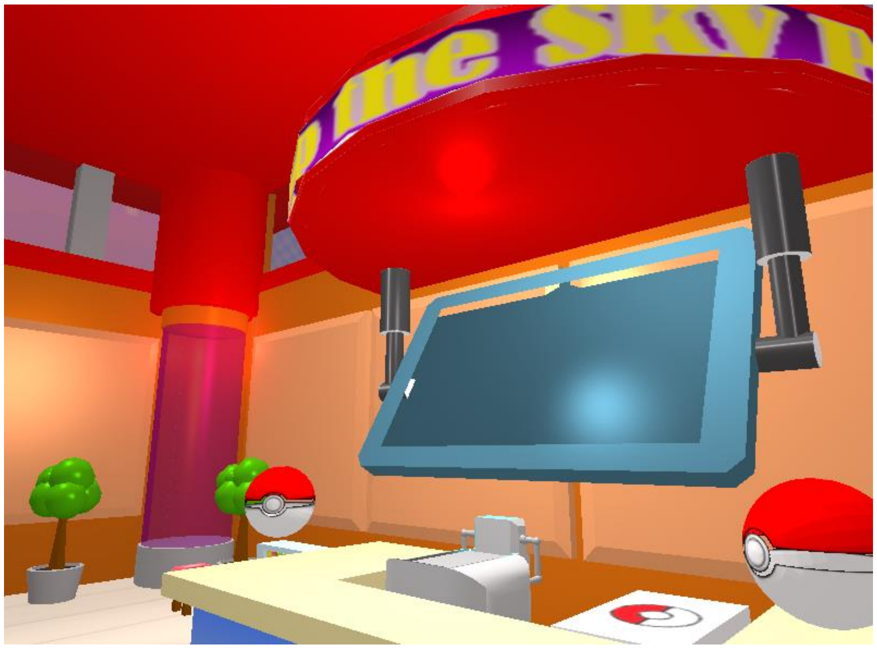
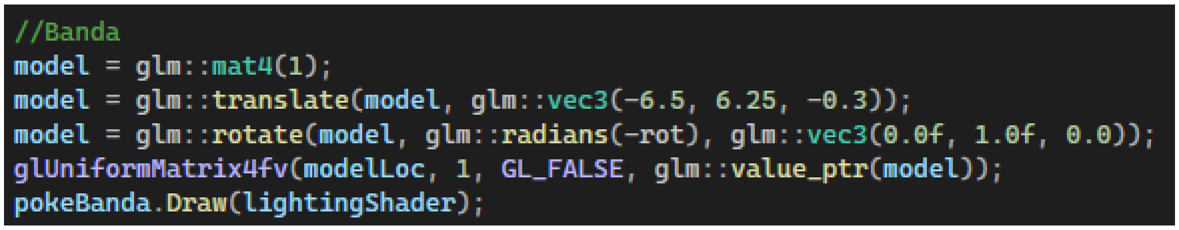


Figure 2.4. TV Animation.

Sign Animation

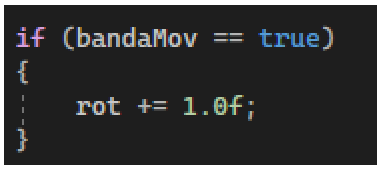


```
//Banda
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-6.5, 6.25, -0.3));
model = glm::rotate(model, glm::radians(-rot), glm::vec3(0.0f, 1.0f, 0.0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
pokeBanda.Draw(lightingShader);
```

Figure 3. Drawing of models within the project and imposition of behaviors.

For the movement of the band-poster that is constantly rotating within the video game, however, for the conditions established for this work, this movement must be activated by a key which starts a rotation on the Y axis that can only be interrupted by pressing the same key once more, but it is necessary that it goes in a reverse rotation to the usual one since by the distribution of the poster it will have to be read from left to right by using the O key.

As with the TV animation, a variable was required to represent the desired rotation for the TV and a second Boolean variable to know when the TV is in motion in order to move to the next increment of rotation one at a time.



```
if (bandaMov == true)
{
    rot += 1.0f;
}
```

Figure 3.1. Function that allows to move the banner band constantly.

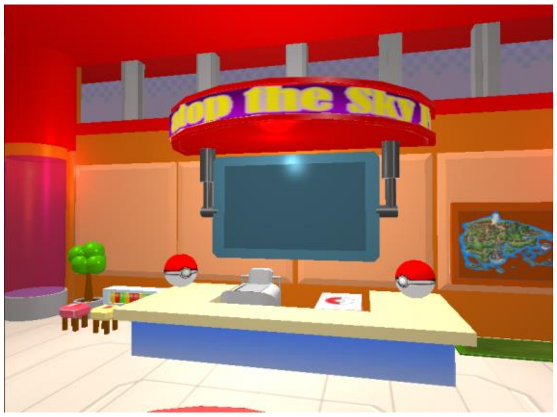
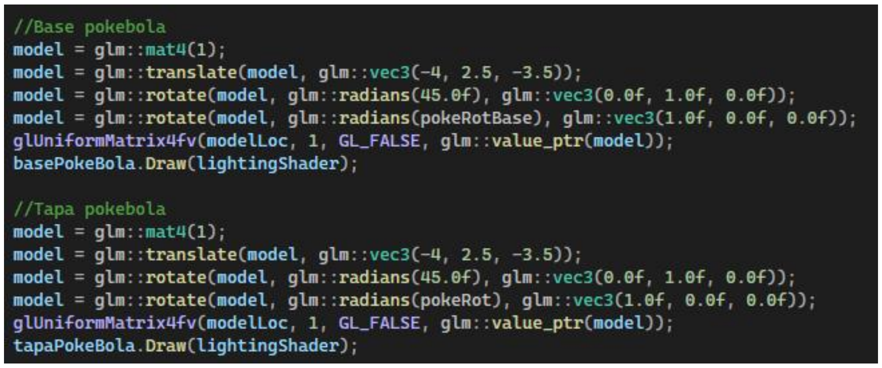


Figure 3.2. Sign Animation.

Pokeballs Animation

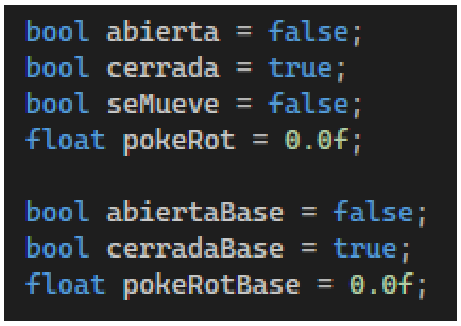
The movement of the pokeballs was intended to resemble the original seen in video games in which both the white and red base open up to a certain angle to be able to deploy one of the monsters inside. In the versions of the pokemon center in Omega Ruby it is common to see on top of the tables a couple of pokeballs that serve to set the scene.



```
//Base pokebola
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-4, 2.5, -3.5));
model = glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(pokeRotBase), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
basePokeBola.Draw(lightingShader);

//Tapa pokebola
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-4, 2.5, -3.5));
model = glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(pokeRot), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
tapaPokeBola.Draw(lightingShader);
```

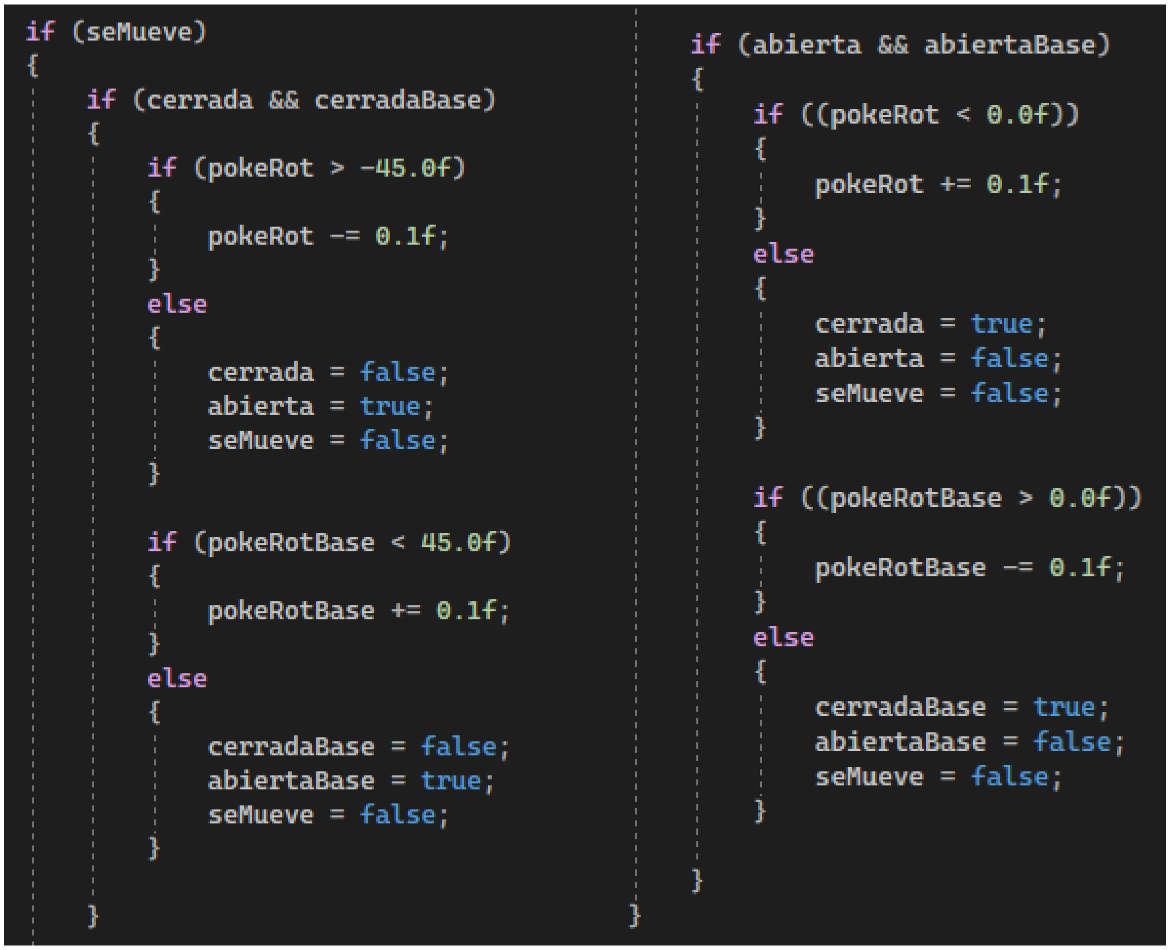
Figure 4. Function that allows to move the banner band constantly..

The animation that allows closing and opening required 5 Boolean variables and two floating variables that could determine the angle over which each of the pokeball lids would rotate using the P key.



```
bool abierta = false;
bool cerrada = true;
bool seMueve = false;
float pokeRot = 0.0f;

bool abiertaBase = false;
bool cerradaBase = true;
float pokeRotBase = 0.0f;
```

Figure 4.1. Variables used for the animation of pokeballs.



```
if (seMueve)
{
    if (cerrada && cerradaBase)
    {
        if (pokeRot > -45.0f)
        {
            pokeRot -= 0.1f;
        }
        else
        {
            cerrada = false;
            abierta = true;
            seMueve = false;
        }

        if (pokeRotBase < 45.0f)
        {
            pokeRotBase += 0.1f;
        }
        else
        {
            cerradaBase = false;
            abiertaBase = true;
            seMueve = false;
        }

    }
```

```
    if (abierta && abiertaBase)
    {
        if ((pokeRot < 0.0f))
        {
            pokeRot += 0.1f;
        }
        else
        {
            cerrada = true;
            abierta = false;
            seMueve = false;
        }

        if ((pokeRotBase > 0.0f))
        {
            pokeRotBase -= 0.1f;
        }
        else
        {
            cerradaBase = true;
            abiertaBase = false;
            seMueve = false;
        }

    }
}
```

Figure 4.2. Function and logic used for the movement of the pokeballs.

In the first instance it is checked if the pokeball is moving, if it is not, then it is checked that both lids are closed, if they are, then the user wants the animation to make the pokeballs open. Both lids will rotate until they reach an angle of 45 degrees, in the case of the lid, the value of the rotation will decrease until it reaches -45 degrees and the base will increase until it reaches the angle. In case both lids are open, then the animation will try to close them and as in the previous case the values of the rotation will be decreasing and increasing depending on the case until both lids are at zero degrees symbolizing the closing of the pokeball.

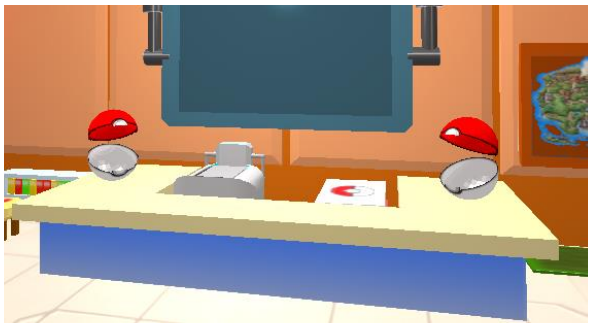


Figure 4.3. Animation of opening of pokeballs.

Figure 4.4. Animation of pokeballs closed animation.

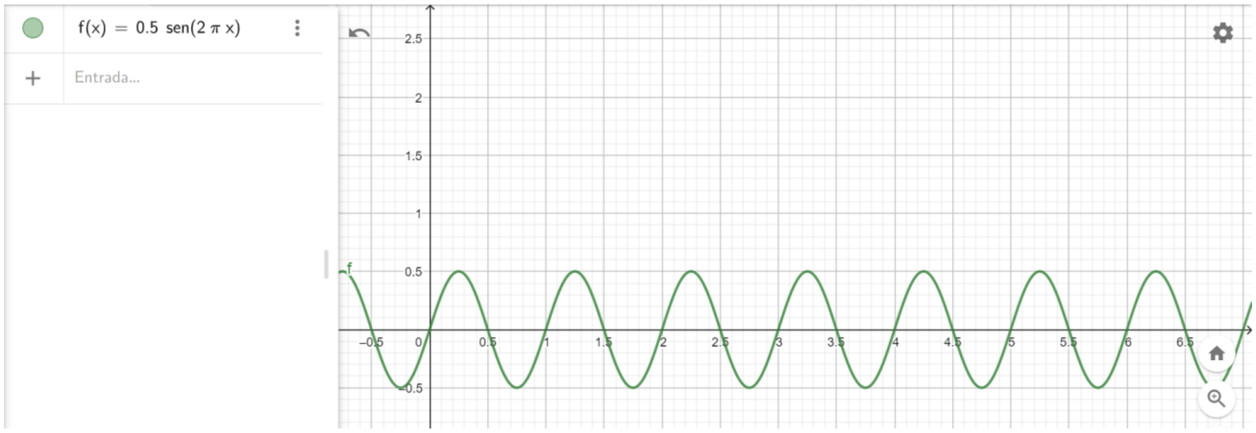
Complex Animations

Bubbles Animation

The idea of the movement of the bubbles was born from the tubes that are on the sides of the pokemon center, in the video game, these bubbles come out at random times, but they are only displaced vertically, however, the movement of the bubbles is usually very erratic and they are not displaced only vertically because at the same time they are oscillating or separating one bubble from another. In the case of the final project, it was decided that the bubbles would oscillate as they moved up the tube.

This oscillation can be represented by a sine function, where the amplitude of the sine function represents the interval over which the bubbles will move from side to side or left to right over a height interval from 0 to 6, which is approximately the height of the tube model.

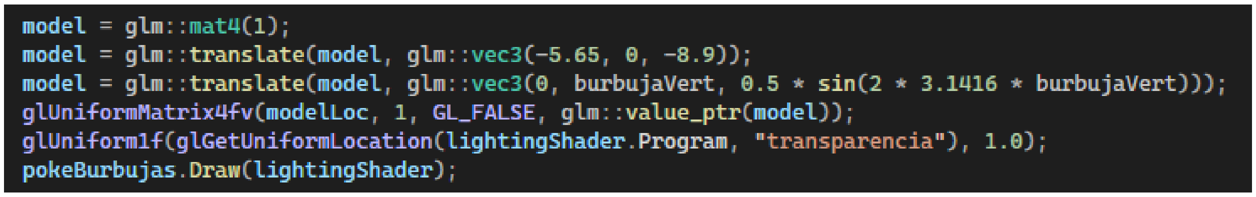
Following then the following function and its representation in Geogebra is how to start its implementation in code.



$f(x) = 0.5\ \mathrm{sen}(2\pi x)$

Figure 5. Function representing the sine oscillation of the bubbles.

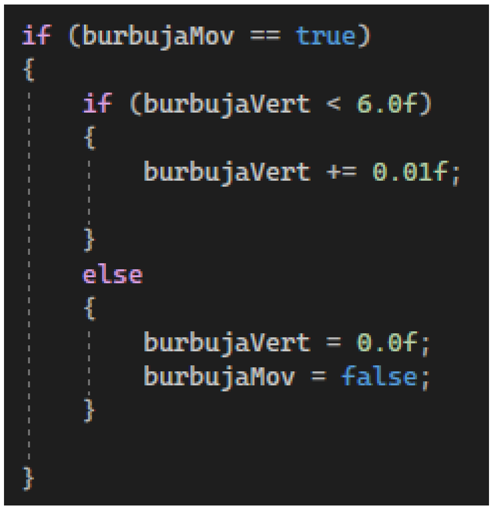
To begin the explanation of the implementation of the bubbles it is important to emphasize that these were one of the translucent objects that exist within the scene, and therefore, require an addition of additional lines of code that allow them to have this visual feature.



```
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-5.65, 0, -8.9));
model = glm::translate(model, glm::vec3(0, burbujaVert, 0.5 * sin(2 * 3.1416 * burbujaVert)));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0);
pokeBurbujas.Draw(lightingShader);
```

Figure 5.1. Drawing and assignment of translucency as property.

Continuing with the animation, a Boolean variable is used to know if the bubbles are in motion and also to allow them to move up as they are oscillating or moving from side to side. The vertical movement is controlled by the *burbujaVert* variable.



```
if (burbujaMov == true)
{
    if (burbujaVert < 6.0f)
    {
        burbujaVert += 0.01f;

    }
    else
    {
        burbujaVert = 0.0f;
        burbujaMov = false;
    }

}
```

Figure 5.2. Function limiting bubble rise.

Which states that when a height of 6 is reached, the animation ends while the oscillation will continue as long as the bubbleVert variable continues to receive values to assign to the function seen in Geogebra with the K key.

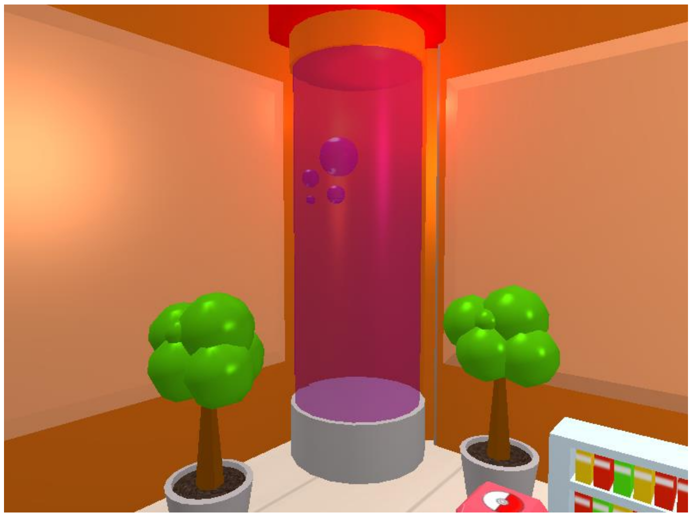
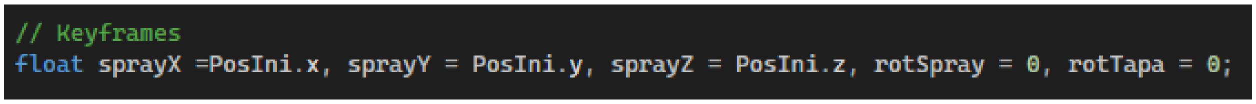


Figura 5.3. Animación de las burbujas.

First Aid Kit Animation

The movement of the first aid kit is an animation made from keyframes, where it is sought that the model of the first aid kit opens its lid and from its interior comes out a first aid spray for the pokemons that go to the pokemon center and is placed on the table for a few seconds and then return inside the first aid kit.

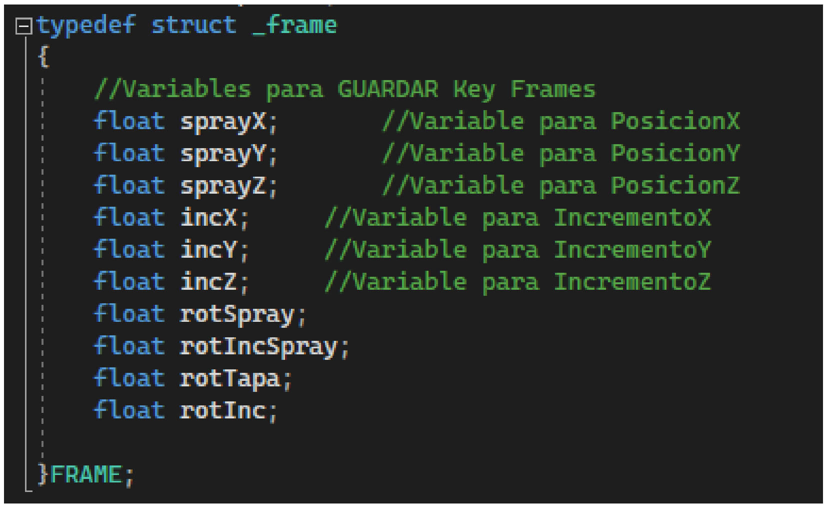
Explained the previous thing, then the lid of the first aid kit will only rotate while the spray will have a displacement and a rotation so that it is put on the table, it is then that the following variables were created that will allow these movements.



```
// Keyframes
float sprayX =PosIni.x, sprayY = PosIni.y, sprayZ = PosIni.z, rotSpray = 0, rotTapa = 0;
```

Figure 6. Variables driving the movements of the first aid kit and spray.

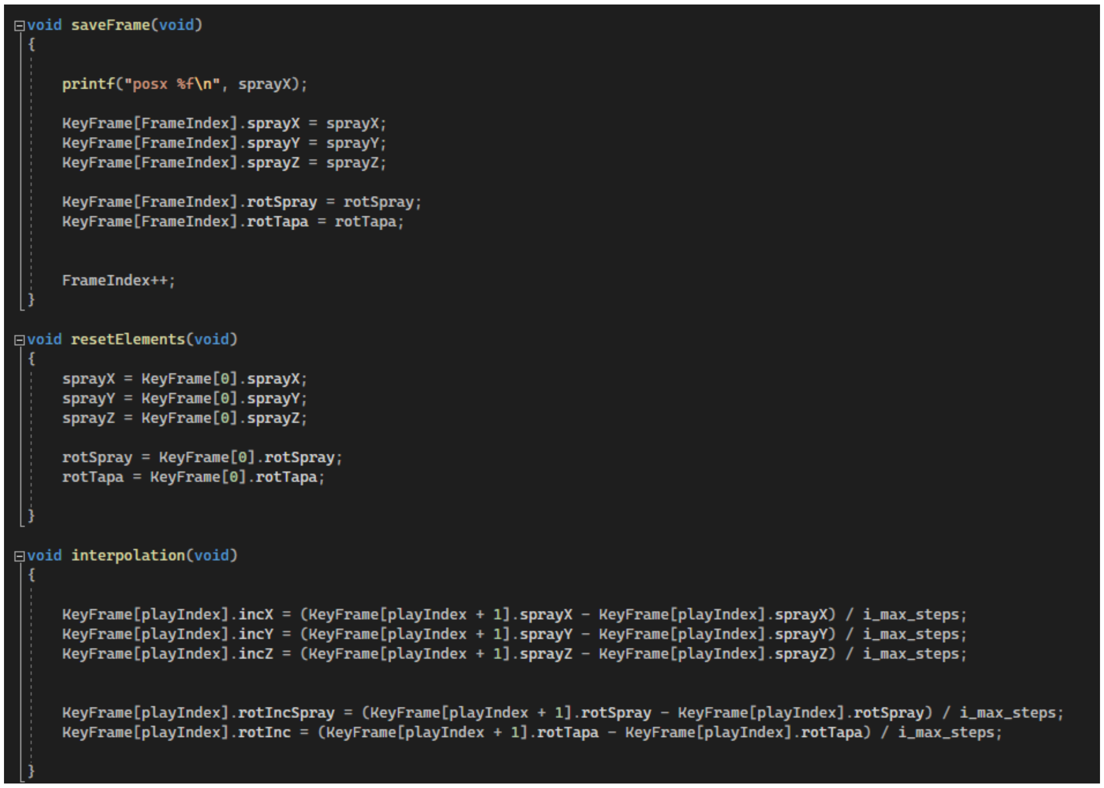
Where the variables *sprayX*, *sprayY* and *sprayZ* are variables that will allow us to manipulate the position of the spray inside the animation while *rotSpray* will allow the handling of its rotation and *rotTapa* that of the first aid kit. Subsequently, we include these variables inside our structure that will capture the frames that we want to be included inside our animation.



```
typedef struct _frame
{
    //Variables para GUARDAR Key Frames
    float sprayX;       //Variable para PosicionX
    float sprayY;       //Variable para PosicionY
    float sprayZ;       //Variable para PosicionZ
    float incX;     //Variable para IncrementoX
    float incY;     //Variable para IncrementoY
    float incZ;     //Variable para IncrementoZ
    float rotSpray;
    float rotIncSpray;
    float rotTapa;
    float rotInc;

}FRAME;
```

Figure 6.1 Inclusion of variables to be captured by the keyframe.

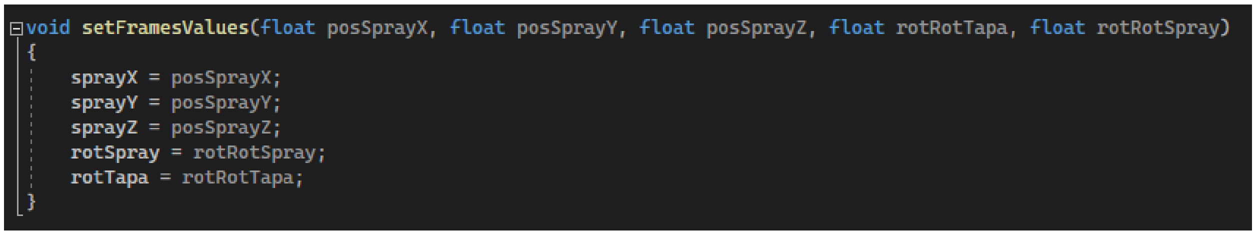
Likewise, since these new variables were created, they also have to be included in the functions that allow to manually take keyframes, these being *saveFrame*(), *resetElements*(), *interpolation*().



```
void saveFrame(void)
{

    printf("posx %f\n", sprayX);

    KeyFrame[FrameIndex].sprayX = sprayX;
    KeyFrame[FrameIndex].sprayY = sprayY;
    KeyFrame[FrameIndex].sprayZ = sprayZ;

    KeyFrame[FrameIndex].rotSpray = rotSpray;
    KeyFrame[FrameIndex].rotTapa = rotTapa;


    FrameIndex++;
}

void resetElements(void)
{
    sprayX = KeyFrame[0].sprayX;
    sprayY = KeyFrame[0].sprayY;
    sprayZ = KeyFrame[0].sprayZ;

    rotSpray = KeyFrame[0].rotSpray;
    rotTapa = KeyFrame[0].rotTapa;

}

void interpolation(void)
{

    KeyFrame[playIndex].incX = (KeyFrame[playIndex + 1].sprayX - KeyFrame[playIndex].sprayX) / i_max_steps;
    KeyFrame[playIndex].incY = (KeyFrame[playIndex + 1].sprayY - KeyFrame[playIndex].sprayY) / i_max_steps;
    KeyFrame[playIndex].incZ = (KeyFrame[playIndex + 1].sprayZ - KeyFrame[playIndex].sprayZ) / i_max_steps;


    KeyFrame[playIndex].rotIncSpray = (KeyFrame[playIndex + 1].rotSpray - KeyFrame[playIndex].rotSpray) / i_max_steps;
    KeyFrame[playIndex].rotInc = (KeyFrame[playIndex + 1].rotTapa - KeyFrame[playIndex].rotTapa) / i_max_steps;

}
```

Figure 6.2 Inclusion of variables to be captured by the keyframe.

In addition to all these functions, one more function was added corresponding to the transfer of information of each of the desired positions and rotations within our animation in order to change the value of the global variables corresponding to these actions.



```
void setFramesValues(float posSprayX, float posSprayY, float posSprayZ, float rotRotTapa, float rotRotSpray)
{
    sprayX = posSprayX;
    sprayY = posSprayY;
    sprayZ = posSprayZ;
    rotSpray = rotRotSpray;
    rotTapa = rotRotTapa;
}
```

Figure 6.3. Handling the value of variables by means of a function.

For the animation, the idea was to open the medicine cabinet and then move the spray slowly, at first, out of the cabinet, and then rotate it so that it is vertically seated on top of the table.

Knowing the above, the following keyframes were used for its realization:

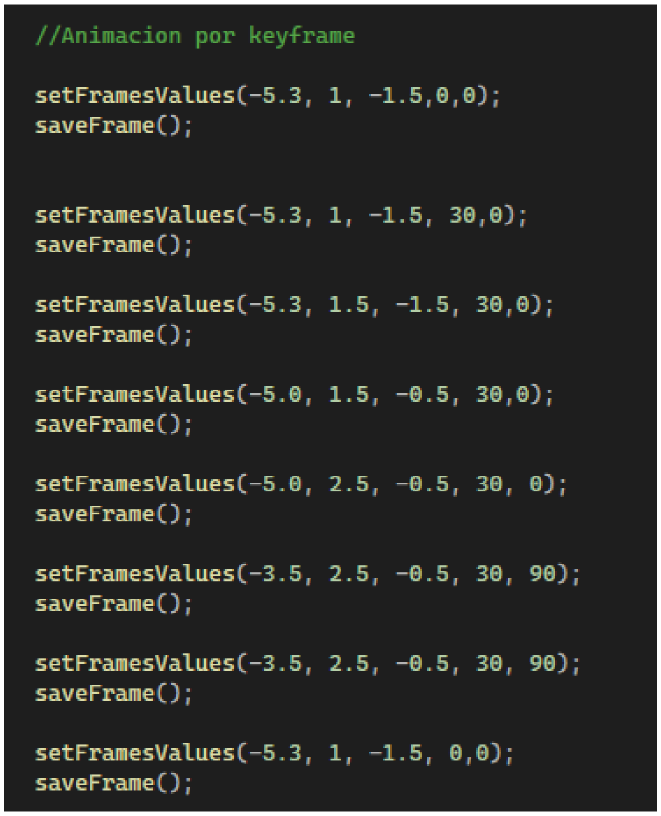
setFramesValues(-5.3, 1, -1.5,0,0); // Initial positions of objects

setFramesValues(-5.3, 1, -1.5, 30,0); // Opening the lid of the first aid kit

setFramesValues(-5.3, 1.5, -1.5, 30,0); // Lift spray inside the first aid kit setFramesValues(-5.0, 1.5, -0.5, 30,0); // Horizontal displacement of the spray out of the kit

setFramesValues(-5.0, 2.5, -0.5, 30, 0); // Lift spray outside the first aid kit

setFramesValues(-3.5, 2.5, -0.5, 30, 90); // Rotating the spray and putting it on the table.



```
//Animacion por keyframe

setFramesValues(-5.3, 1, -1.5,0,0);
saveFrame();


setFramesValues(-5.3, 1, -1.5, 30,0);
saveFrame();

setFramesValues(-5.3, 1.5, -1.5, 30,0);
saveFrame();

setFramesValues(-5.0, 1.5, -0.5, 30,0);
saveFrame();

setFramesValues(-5.0, 2.5, -0.5, 30, 0);
saveFrame();

setFramesValues(-3.5, 2.5, -0.5, 30, 90);
saveFrame();

setFramesValues(-3.5, 2.5, -0.5, 30, 90);
saveFrame();

setFramesValues(-5.3, 1, -1.5, 0,0);
saveFrame();
```

Figure 6.4 Keyframes taken with the positions of our animation.

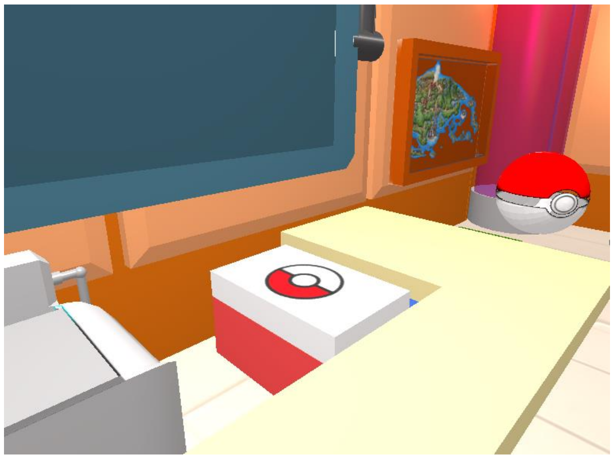


Figure 6.5 First keyframe with initial positions.

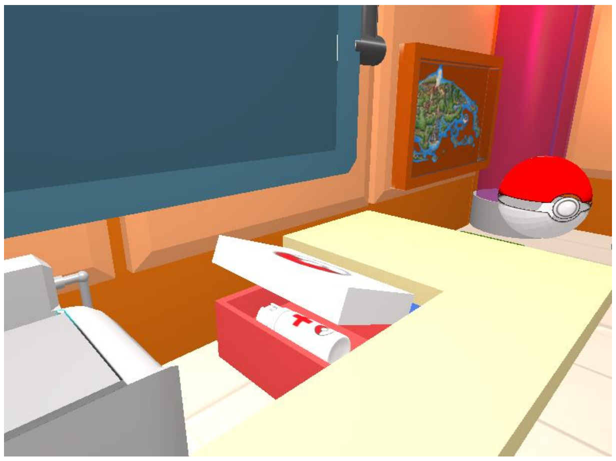


Figure 6.6 Second keyframe with lid open.

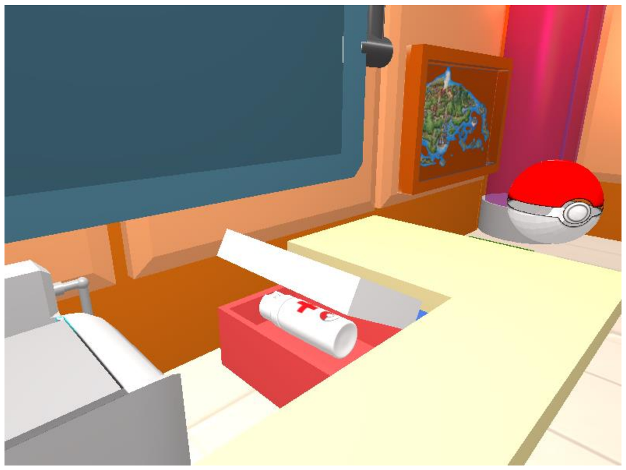


Figure 6.7 Third keyframe with cover open.

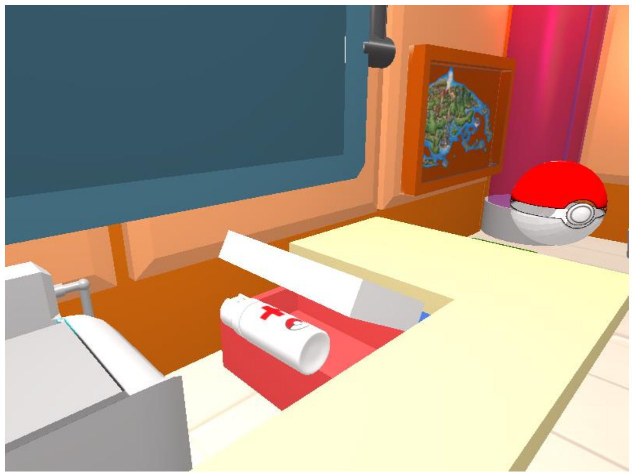


Figure 6.8 Fourth keyframe with the spray out of the first aid kit.

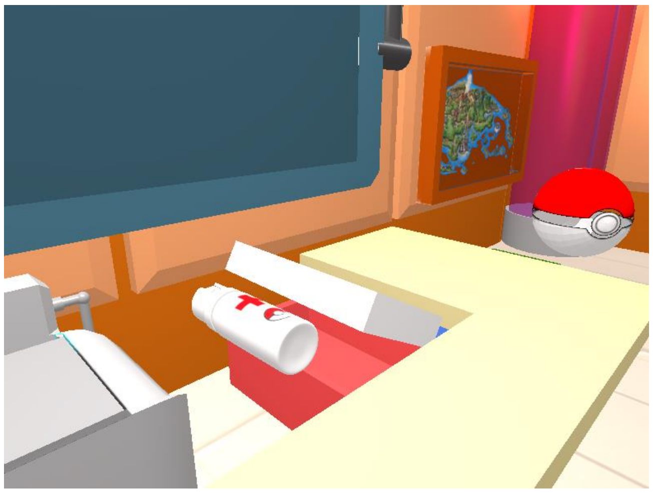


Figure 6.9 Fifth keyframe sprayer lift out of first aid kit.

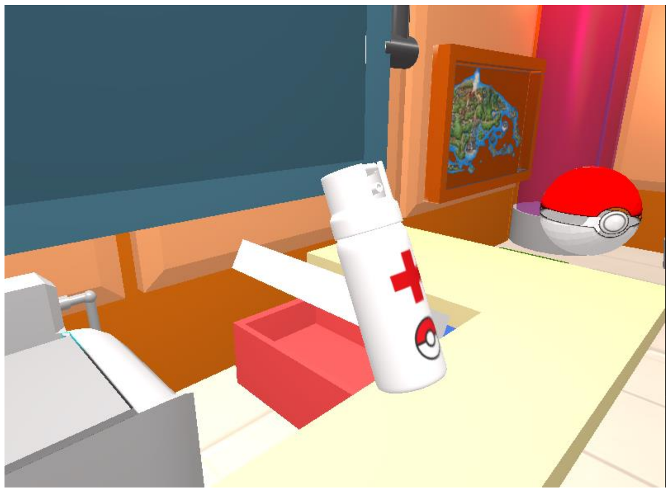


Figure 6.10 Sixth keyframe setting of the spray on the table.

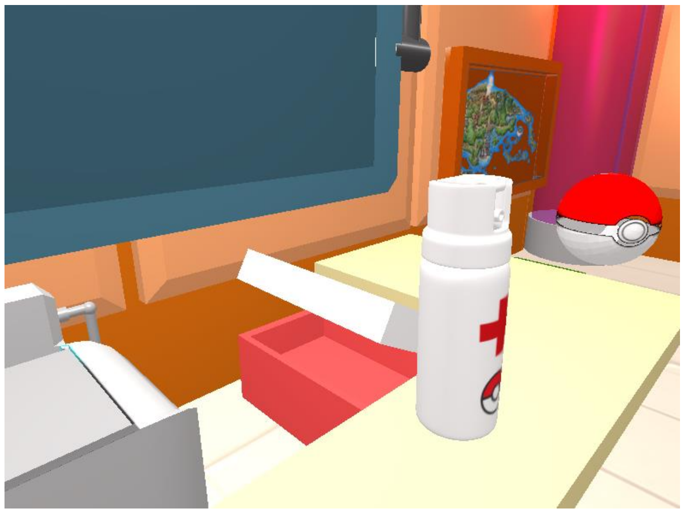
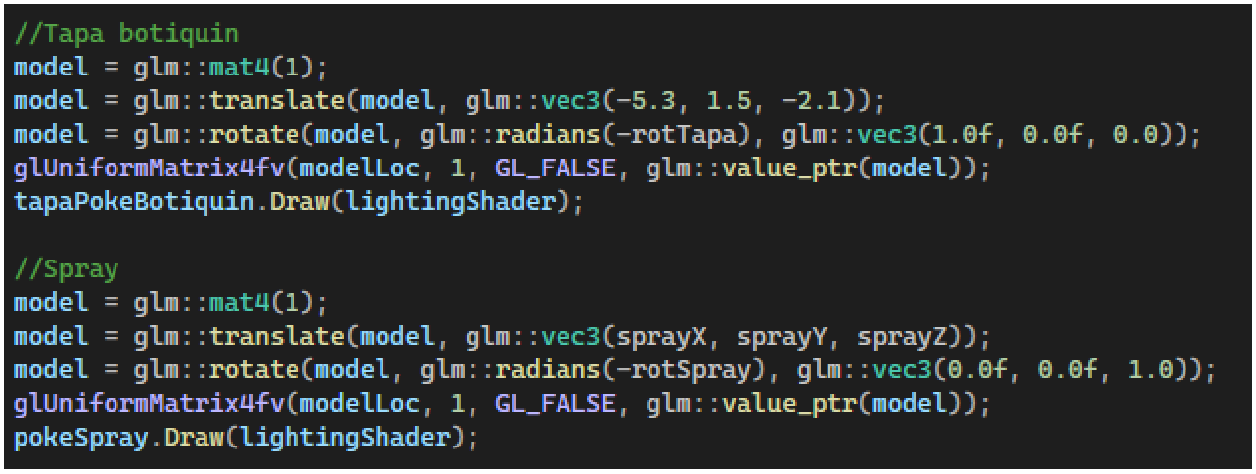


Figure 6.11 Seventh and Eighth keyframe of the spray on the table.

Once this is done, the only thing left to do is to control the model of the first-aid kit and the spray by means of the L key.

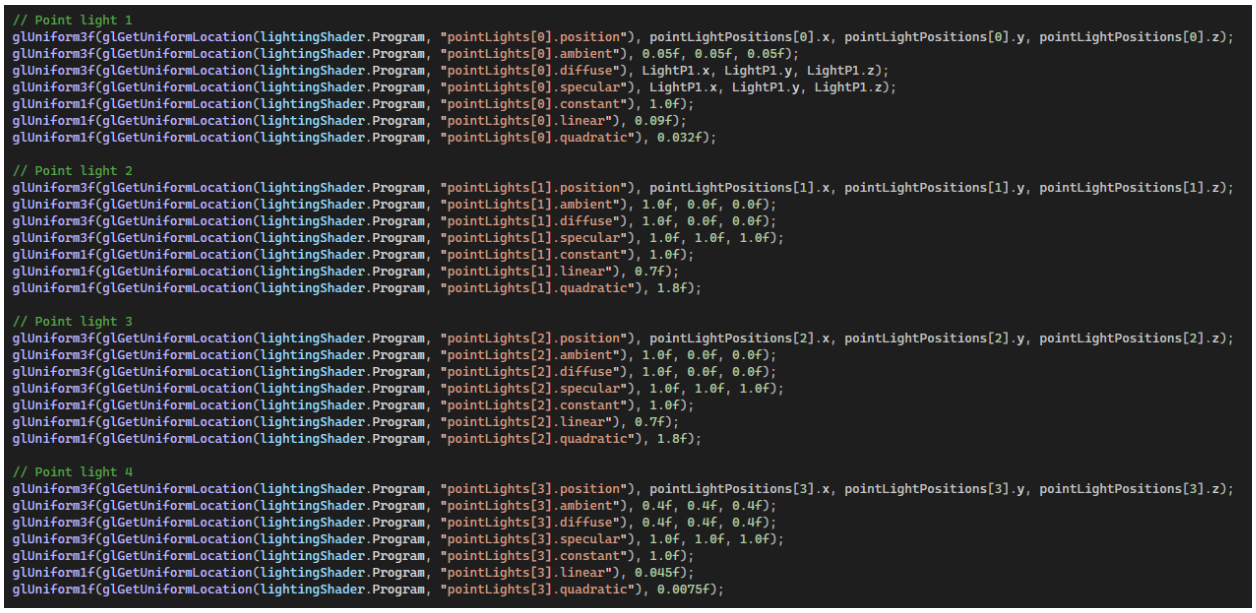


```
//Tapa botiquin
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-5.3, 1.5, -2.1));
model = glm::rotate(model, glm::radians(-rotTapa), glm::vec3(1.0f, 0.0f, 0.0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
tapaPokeBotiquin.Draw(lightingShader);

//Spray
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(sprayX, sprayY, sprayZ));
model = glm::rotate(model, glm::radians(-rotSpray), glm::vec3(0.0f, 0.0f, 1.0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
pokeSpray.Draw(lightingShader);
```

Figure 6.12 Handling OpenGL transformations using our keyframe-controlled variables.

Lights

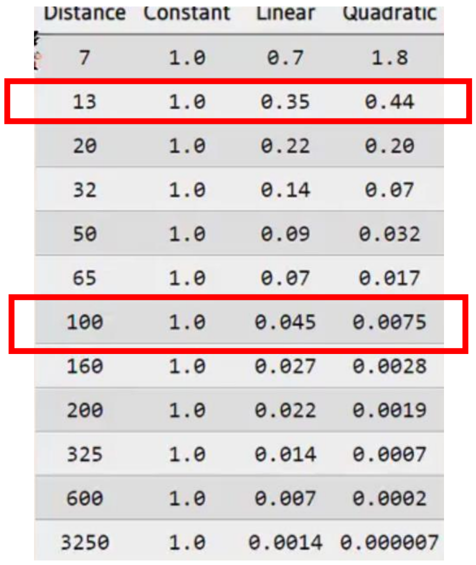
As the last part of this technical manual, mention will be made of the lights that were used to set the scene with a total of four lights.



```
// Point light 1
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), 0.05f, 0.05f, 0.05f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), LightP1.x, LightP1.y, LightP1.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), LightP1.x, LightP1.y, LightP1.z);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 0.032f);

// Point light 2
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].position"), pointLightPositions[1].x, pointLightPositions[1].y, pointLightPositions[1].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].ambient"), 1.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].diffuse"), 1.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].linear"), 0.7f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].quadratic"), 1.8f);

// Point light 3
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].position"), pointLightPositions[2].x, pointLightPositions[2].y, pointLightPositions[2].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].ambient"), 1.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].diffuse"), 1.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].linear"), 0.7f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].quadratic"), 1.8f);

// Point light 4
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].position"), pointLightPositions[3].x, pointLightPositions[3].y, pointLightPositions[3].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].ambient"), 0.4f, 0.4f, 0.4f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].diffuse"), 0.4f, 0.4f, 0.4f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].linear"), 0.045f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].quadratic"), 0.0075f);
```

Figure 7 Definition of stage lights

The lights used are located at 4 specific points within the structure, one under the circular sign in the center, one on the ceiling and the other two inside each of the tubes to illuminate the bubbles, the latter two being the ones with the lowest dispersion due to the smaller area they have to illuminate and the one on the ceiling, the one with the highest dispersion, being aware that this light is responsible for illuminating the entire pokemon center. To achieve these effects then values were assigned to the linear and quadratic quantities of the lights based on the following table.



Distance	Constant	Linear	Quadratic
7	1.0	0.7	1.8
13	1.0	0.35	0.44
20	1.0	0.22	0.20
32	1.0	0.14	0.07
50	1.0	0.09	0.032
65	1.0	0.07	0.017
100	1.0	0.045	0.0075
160	1.0	0.027	0.0028
200	1.0	0.022	0.0019
325	1.0	0.014	0.0007
600	1.0	0.007	0.0002
3250	1.0	0.0014	0.000007

Conclusion

With the realization of the present work it was possible to put into practice each and every one of the concepts seen in the course to be able to adequately handle each of the concepts of OpenGL and to know its scope and limitations as well as the possibilities that a student initiated in the area has to represent projects from different perspectives, hoping that some of the knowledge obtained in the subject will be useful in the labor field.