



**UNIVERSIDAD NACIONAL AUTÓNOMA  
DE MÉXICO**

---

---

**FACULTAD DE INGENIERÍA**

**(Recreación 3D de la casa de los McCallister de Home  
Alone)**

**MANUAL TÉCNICO**

**(Ramírez Rivera Giezy Alberto)**

**FECHA DE ENTREGA : 19 de enero de 2021**



## ÍNDICE

### Contenido

Objetivos .....	3
Alcance.....	4
Actividades.....	5
Documentación del código.....	6
<b>Key Frames</b> .....	6
<b>Animación 1</b> .....	7
<b>Animaciones simples</b> .....	10
<b>Main</b> .....	12
Créditos y referencias.....	14

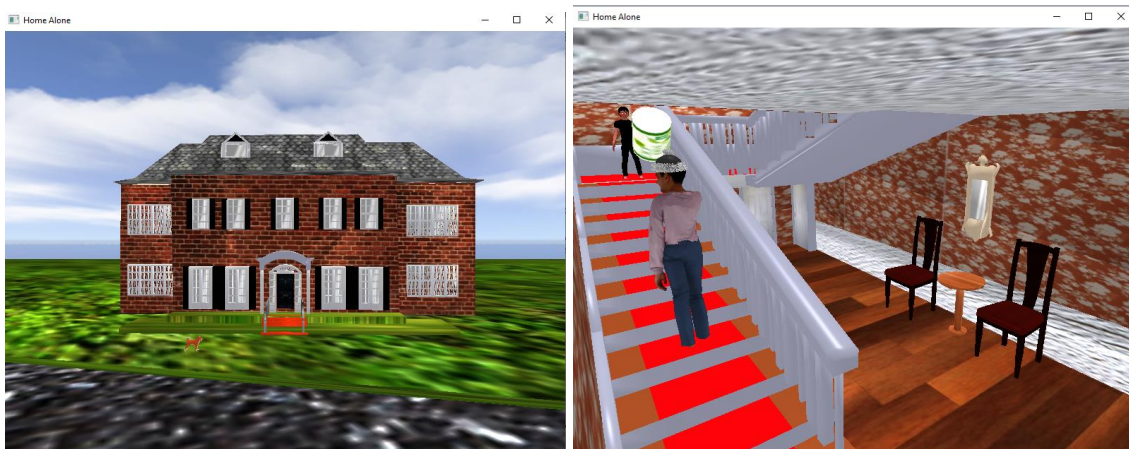
## Objetivos

Desarrollar un espacio virtual que simule la casa de “mi pobre angelito”.



La casa tendrá muebles en los primeros cuartos. Se mostrarán sillas, mesas, macetas con planta, un espejo, un sillón y algunos otros muebles.

Además, se contarán con dos animaciones complejas y tres sencillas. Para las animaciones complejas, se utilizará la estrategia de key frames.



## Alcance

Sobre la fachada: La casa debe parecerse a la casa original en la parte exterior, presentando las mismas ventanas, la puerta y las decoraciones. Las plantas de la fachada pueden no existir debido a la complejidad de modelar plantas.



El interior de la casa debe consistir en los dos primeros pisos de la casa, aunque sólo los primeros cuartos tendrán los muebles correspondientes. En el interior se modelan los siguientes objetos:


















- Silla color rojo (múltiples instancias)
- Espejo
- Maceta con planta roja
- Sillón personal color anaranjado
- Mesa grande (para el comedor)
- Mesas pequeñas
- Cajonera
- Mueble con cajones y puertas (modelado con primitivas básicas)
- Persona (ladrón)
- Kevin
- Bote de pintura colgado al techo
- Araña
- Perro





El proyecto también debe contener 5 animaciones, de las cuales 2 son hechas con la técnica de key frames. Las animaciones complejas son: caída del ladrón después de ser golpeado por el bote de pintura y Kevin subiendo las escaleras. Las animaciones simples son: recorrido de un perro, recorrido de una araña y apertura de la puerta de entrada.

## Actividades

id		Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	ct '20 04   11   18   25	nov '20 01   08   15   22   29	dic '20 06   13   20   27	ene '21 03   10   17   24
1			<b>Proyecto de Lab. CG. Recreación de casa 'Home Alone'</b>	65 días?	mié 21/10/20	mar 19/01/21					
2			Análisis de requerimientos	4 días	mié 21/10/20	lun 26/10/20					
3			<b>Modelado de fachada</b>	55 días	mié 21/10/20	mar 05/01/21					
7			<b>Modelos 3D adicionales</b>	49 días	mié 21/10/20	lun 28/12/20					
10			Animación	10 días	vie 01/01/21	jue 14/01/21					
11			Manuales	10 días	vie 01/01/21	jue 14/01/21					
12											

## Documentación del código

### Key Frames

Para realizar las animaciones por key frames, se cuenta con dos estructuras: `_frame` y `_frame2`. Estas estructuras definen las variables que se usan en cada key frame para realizar las animaciones.

```
typedef struct _frame
{
    //Variables para GUARDAR Key Frames
    float posX;           //Posición en x del ladrón
    float posY;           //Posición en y del ladrón
    float posZ;           //Posición en z del ladrón
    float incX;           //incrementos
    float incY;           //
    float incZ;           //
    float rotRodIzq;      //Rotación de la rodilla izquierda
    float rotRodDer;      //Rotación de la rodilla derecha
    float rotCodIzq;      //Rotación del codo izquierdo
    float rotCodDer;      //Rotación del codo derecho
    float rotTorso;       //Rotación general (torso)
    float rotNeck;        //Rotación del cuello
    float rotInc;         //incrementos de las rotaciones
    float rotInc2;
    float rotInc3;
    float rotInc4;
    float rotInc5;
    float rotInc6;
}FRAME;

typedef struct _frame2
{
    //Variables para GUARDAR Key Frames
    float posX;           //posición en x de Kevin
    float posY;           // posición en y de Kevin
    float posZ;           // posición en z de Kevin
    float incX;           // incrementos
    float incY;           //
    float incZ;           //
    float rotBoy;         //Rotación del chico en y
    float rotInc;         //incremento de rotación
}FRAME2;
```

Después de que se definen las variables, se tienen ciertas funciones. `resetElements()` y `resetElements2()` se encargan de iniciar las variables en su valor en el key frame 0, permitiendo que las animaciones puedan reiniciarse. Estas funciones no tienen parámetros.

Se tienen también las funciones de interpolación `interpolation()` e `interpolation2()` que calculan los puntos intermedios entre cada key frame de ambas animaciones. La variable `i_max_steps` define cuántos pasos o puntos intermedios tendrá cada interpolación.

La función `keyCallback(GLFWwindow *window, int key, int scancode, int action, int mode)` identifica cuando el usuario presiona una tecla. Al presionar la tecla '1', se inicia la primera animación; consecutivamente, ocurre lo mismo para las teclas '2' y '3'. Además de esto, revisa cuando el usuario presiona la tecla 'escape', cerrando la aplicación. Esta función, para el caso de las tres animaciones, hace uso de variables booleanas para avisar a la función `animación()` que debe reproducir una u otra animación.

Finalmente, la función `animación()` detecta cuando una animación debe iniciarse, realiza los pasos previos a la animación correspondiente (que podría ser resetear variables o iniciar una animación previa, como en el caso de la animación '1'), llama a `resetElements()` y llama a la función `interpolación()` para cada frame hasta terminar la animación correspondiente.

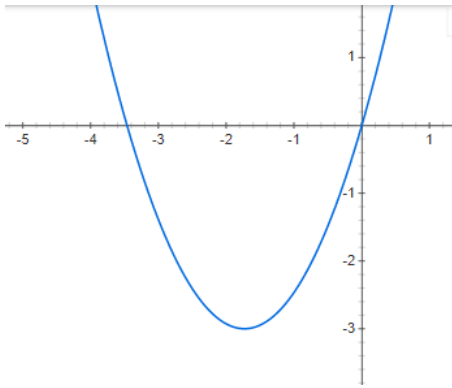
## Animación 1

Al iniciarse la animación (con la tecla '1'), se resetean las variables del bote de pintura y se cambia la variable `pPlay=true` para iniciar la animación del bote.

```
if (pPlay == false && (FrameIndex > 1))
```

```
    {
        Ypaint = 0;
        Zpaint = 0;
        rotPaint = 0;
        pPlay = true;
    }
    else
    {
        play = false;
        //pPlay = false;
    }
```

Cuando esto ocurre, `animación()` detecta que debe iniciar la animación del bote. El movimiento es controlado por una parábola de función  $y = (x + 1.73)^2 - 3.0$



Puesto que la parábola inicia con  $x=0$ , al decrementarse  $x$ , y se moverá de forma parabólica, generando el efecto de columpio deseado. Además, la rotación decrementa en 4.5 en cada paso (cada paso,  $x$  decremента en 0.2). Nótese que, para este caso, es necesario intercambiar  $X$  por  $Z$  por la posición del bote de pintura. Cuando termina la animación del bote, continúa la animación del ladrón por key frames, iniciada con el booleano 'play'.

```
if (pPlay)
{
    if (Zpaint >= -4.0f) {
        Zpaint -= 0.2f;
        Ypaint = (Zpaint + 1.73f)*(Zpaint + 1.73f) - 3.0f;
        rotPaint -= 4.5f;
    }
    else {
        resetElements();
        //First Interpolation
        interpolation();

        play = true;
        playIndex = 0;
        i_curr_steps = 0;

        pPlay = false;
    }
}
```





Al iniciarse 'play', se repite el proceso de interpolado hasta llegar a `i_max_steps`. A su vez, este proceso se repite hasta que `playIndex` llegue al máximo de frames. En cada paso, se actualizan las variables que manipulan el movimiento de los objetos.

```

if (play)
{
    //printf("reproduciendo");
    if (i_curr_steps >= i_max_steps) //end of animation between frames?
    {
        playIndex++;
        if (playIndex > FrameIndex - 2)    //end of total animation?
        {
            printf("termina anim\n");
            playIndex = 0;
            play = false;
        }
        else //Next frame interpolations
        {
            i_curr_steps = 0; //Reset counter
                                   //Interpolation
            interpolation();
        }
    }
    else
    {
        //Draw animation

        posX += KeyFrame[playIndex].incX;
        posY += KeyFrame[playIndex].incY;
        posZ += KeyFrame[playIndex].incZ;

        rotRodIzq += KeyFrame[playIndex].rotInc;
        rotRodDer += KeyFrame[playIndex].rotInc2;

        rotCodIzq += KeyFrame[playIndex].rotInc3;
    }
}

```

```

        rotCodDer += KeyFrame[playIndex].rotInc4;

        rotTorso += KeyFrame[playIndex].rotInc5;

        rotNeck += KeyFrame[playIndex].rotInc6;

        i_curr_steps++;
    }
}

```

La tercera animación funciona de la misma forma, quitando la parte del bote de pintura.



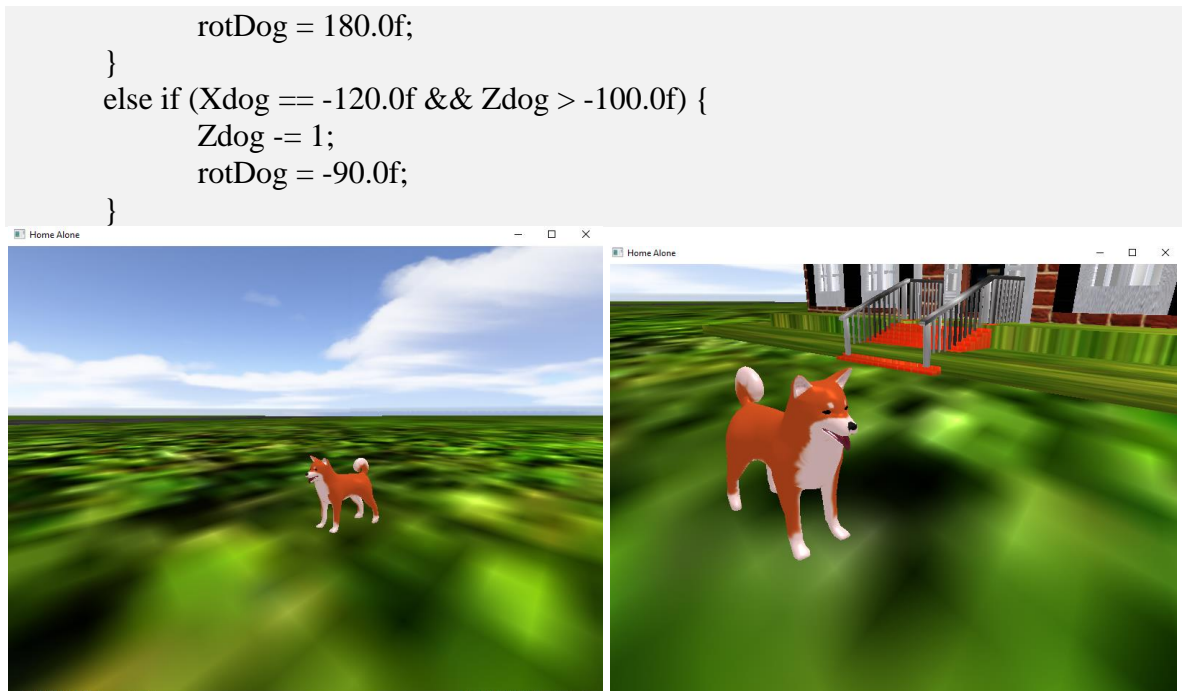
### Animaciones simples

La animación del perro funciona de una manera muy simple. Dentro de la función `animación()` se tiene una sección de código que se ejecuta siempre. Con condiciones tipo `if`, la función detecta en qué parte del recorrido se encuentra el perro y actualiza la variable correspondiente hasta que cambie de sección. Por ejemplo, si se encuentre en la parte frontal de la casa, significa que `Zdog = -100` y `Xdog < 140`; mientras esté en esta sección, `X` se suma. Una vez llegue a la posición `Xdog=140`, se sabe que el perro ha llegado al lado derecho de la casa, por lo que cambia su ruta y comienza a disminuirse `Zdog`.

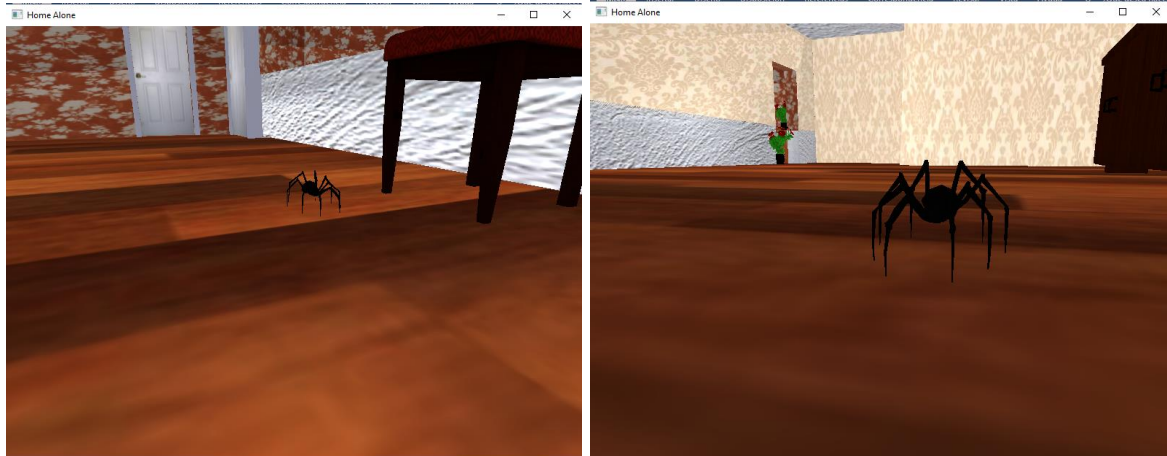
```

//movimiento del perro
if (Zdog == -100.0f && Xdog < 140.0f) {
    Xdog += 1.0f;
    rotDog = 0.0f;
}
else if (Xdog == 140.0f && Zdog < 140.0f) {
    Zdog += 1;
    rotDog = 90.0f;
}
else if (Zdog == 140.0f && Xdog > -120.0f) {
    Xdog -= 1;
}

```



La animación de la araña funciona de la misma forma, aunque con otras coordenadas.



Finalmente se tiene la animación activada con la tecla '2'. Al activarse, sencillamente se reinicia la rotación de la puerta y se activa el booleano de la animación.

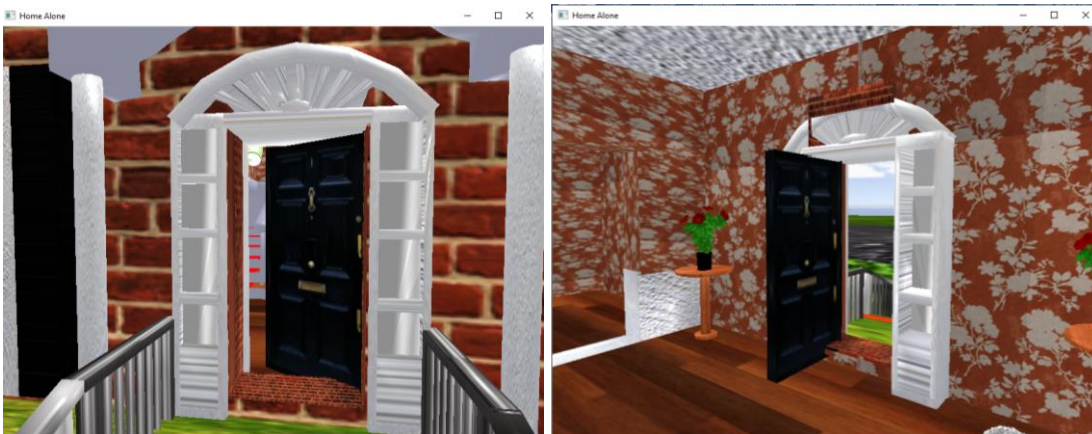
```

if (keys[GLFW_KEY_2])
{
    //printf("animacion 2");
    if (play2 == false) {
        //printf("play 2 es false");
        rotDoor = 0.0f;
        play2 = true;
    }
}

```

Dentro de `animación()`, sencillamente se comprueba que la rotación haya llegado a  $-90^\circ$ ; mientras esto no se cumpla, se va restando la rotación en  $1^\circ$ .

```
if (play2) {
    //printf("play2 ya es true");
    if (rotDoor > -90.0f) {
        rotDoor -= 1.0f;
    }
    else {
        play2 = false;
    }
}
```



## Main

Entre algunas cosas importantes de hacer notar del código, está la inicialización de cada key frame para las animaciones, que se realiza de la siguiente forma:

```
//animación del ladron
KeyFrame[0].posX = 0.0f;
KeyFrame[0].posY = 0.0f;
KeyFrame[0].posZ = 0.0f;
```

De esta forma, se tienen las coordenadas para cada frame.

Después de los frames, se tienen las cargas de las texturas para la skybox y, inmediatamente después, las dos texturas necesarias para cargar el objeto diseñado con primitivas básicas. Cabe mencionar que cada textura usa variables diferentes para no sobrescribirlas.

```
GLuint texture1, texture2;
glGenTextures(1, &texture1);
glGenTextures(1, &texture2);
//...
```

```

GLuint texture3, texture4;
    glGenTextures(1, &texture3);
    glGenTextures(1, &texture4);

```

A continuación, lo primero en dibujarse es el mueble con primitivas básicas. Para esto, se usó una estrategia en donde se dibujaron las dos patas y la parte principal del cuerpo, pero ésta sin la primera de sus caras (la frontal); una vez dibujados estos objetos con la primera textura, se dibuja la cara restante con la segunda textura, generando el modelo completo.

```

// Bind diffuse map
    glBindTexture(GL_TEXTURE_2D, texture1);

    // Bind specular map
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture2);

    glBindVertexArray(VAO);
    glm::mat4 tmp = glm::mat4(1.0f); //Temp
    glm::mat4 model(1);

    model = glm::mat4(1);
    model = glm::translate(model, glm::vec3(6.0f, -7.5f, 130.0f));
    model = glm::scale(model, glm::vec3(3.0f, 5.0f, 5.0f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    glDrawArrays(GL_TRIANGLES, 0, 36);

    model = glm::mat4(1);
    model = glm::translate(model, glm::vec3(-6.0f, -7.5f, 130.0f));
    model = glm::scale(model, glm::vec3(3.0f, 5.0f, 5.0f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    glDrawArrays(GL_TRIANGLES, 0, 36);

    model = glm::mat4(1);
    model = glm::translate(model, glm::vec3(0.0f, 0.0f, 130.0f));
    model = glm::scale(model, glm::vec3(15.0f, 10.0f, 5.0f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    glDrawArrays(GL_TRIANGLES, 6, 36);

    // Bind diffuse map
    glBindTexture(GL_TEXTURE_2D, texture3);

    // Bind specular map
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture4);

```

```
glBindVertexArray(VAO);

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 6);
```

Finalmente, se dibujan todos los modelos cargados. La mayoría de modelos fueron diseñados con la posición y escala bien definidos en un programa externo, por lo que no fue necesario realizar escalas, traslaciones ni rotaciones. Los modelos a animar sí tienen estas transformaciones, que dependen de las variables antes definidas para determinar su posición, por ejemplo:

```
view = camera.GetViewMatrix();
    model = glm::mat4(1);
    model = glm::translate(model, glm::vec3(Zdog, -6.0f, Xdog));
    model = glm::rotate(model, glm::radians(rotDog), glm::vec3(0.0f, 1.0f,
0.0f));

    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    Dog.Draw(lightingShader);
```

## Créditos y referencias

Modelos creados por:

- Khalidsrri – buró
- orii - flores hibiscos
- numikPopulate – hombre
- igorbob – niño
- jack-pan1972 - perro shiba inu
- lyrog – araña

Todos los modelos tienen licencia Royalty Free. Algunos modelos fueron modificados y/o retexturizados con motivo de que combinaran mejor en el ambiente conforme al contexto del proyecto.

Los modelos no mencionados fueron creados por completo por el autor Giezy Alberto Ramírez Rivera. Toda modificación o creación de modelos fue realizada en 3Ds max 2021. Esqueleto del código original perteneciente al Ing. Carlos Aldair Roman Balbuena.