MANUAL TÉCNICO

OBJETIVO

El alumno deberá aplicar y demostrar los conocimientos adquiridos durante el curso mediante la selección de un espacio tridimensional, donde se pretende recrear virtualmente el realismo de los objetos incluidos en dicho espacio.

DESCRIPCIÓN DEL PROYECTO

En el presente proyecto, se pretendía recrear el entorno basado en el juego de Rockstar Games, GTA San Andreas. En éste, debería haber un mínimo de 7 objetos con los que pudiera existir alguna interacción entre el usuario y el objeto. Para ello, se contaba con la siguiente planificación:

	Semanas						
Actividades	**1 7 – 13 de diciembre**	**2 14 – 20 de diciembre**	**3 21 – 27 de diciembre**	**4 28 – 3 de enero**	**5 4 – 10 de enero**	**6 11 – 17 de enero**	**7 18 – 22 de enero**
Primeros bosquejos de los objetos a implementar	**X**						
Capacitación en el manejo de software de modelado.		**X**					
Modelado de la fachada del inmueble			**X**				
Modelado de los primeros elementos interiores.				**X**			
Texturización de los elementos interiores.					**X**		
Carga en OpenGL con los elementos disponibles.						**X**	
Modelado de últimas figuras y programación de animaciones							**X**

Desafortunadamente, la última actividad planeada no pudo llevarse a cabo de forma satisfactoria, pues las dificultades técnicas en la implementación de los últimos modelos provocaron grandes retrasos en la entrega del entorno, especialmente en la última semana. Entre estas dificultades, se pueden destacar las constantes llamadas a Excepciones donde se especificaba que los vectores normales de las mallas y objetos no podían ser procesadas correctamente.

Cabe destacar que estos mensajes aparecieron el último día previo a la entrega del proyecto, por lo que no fue previsto con anticipación. Estos acontecimientos fueron causados por el intento de implementación de materiales translúcidos. Dado este intento fallido, la transmisión materiales y objetos cuyas propiedades ya se habían logrado apreciar claramente en la ejecución del programa en días anteriores ya no fue permitida. Por lo que para intentar solucionar lo anterior, se trató de reutilizar el modelado geométrico de días anteriores, creyendo que se podían lograr los ajustes necesarios para recuperar los modelos agregados.

Pero, en general, se utilizó el siguiente código para cargar los modelos:

```
Model cjHouse2((char*)"Models/cjHouse4/cjHouse_rec.obj");
//Model cjHouse2((char*)"Models/cjHouseFinal/cjHouseFinal.obj");
Model Carroseria((char*)"Models/Chrysler/chrysler.obj");
Model LlantaIzqSup((char*)"Models/Chrysler/llantaSupIzq.obj");
Model LlantaDerSup((char*)"Models/Chrysler/llantaSupDer.obj");
Model LlantaIzqInf((char*)"Models/Chrysler/llantaInfIzq.obj");
Model LlantaDerInf((char*)"Models/Chrysler/llantaInfDer.obj");
Model PuertaPrinc((char*)"Models/Puerta/puertaPrinc.obj");
```

El archivo “cjHouseFinal.obj” es el que suele mandar excepciones, así que se optó por una versión anterior.

Cada modelo en OpenGL suele tener esta configuración:

```
/*Casa*/
view = camera.GetViewMatrix();
glm::mat4 model(1);
model = glm::mat4(1);
//model = glm::translate(model, glm::vec3(-2.5f, -3.2f, -0.6f));
//model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.6f, 0.6f, 0.6f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
cjHouse2.Draw(lightingShader);

//Puerta principal
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::scale(model, glm::vec3(0.6f, 0.6f, 0.6f));
model = glm::translate(model, glm::vec3(-3.6f, 0.0f, 6.8));
model = glm::rotate(model, glm::radians(rotPuerta), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(3.6f, 0.0f, -6.8f));
PuertaPrinc.Draw(lightingShader);
```

Se observa que, por cada modelo, es importante asignarle la matriz de modelo con valores inicializados en 1. Este paso es importante, pues permite que se conserven las coordenadas del mundo que fueron asignadas en el software de modelado.

Para el caso del auto, era importante dividir los componentes de éste en 5 partes: La propia carrocería y las 4 ruedas. El objetivo con esta separación era obtener una animación donde se pudiera aplicar el modelado jerárquico: El movimiento de las ruedas produciría el movimiento del material.

También, desafortunadamente, la animación no fue obtenida satisfactoriamente, pues el programa no detectaba con claridad que el modelo tenía que moverse junto a la rotación. La misma situación se presentó con la puerta principal del objeto. Los *callbacks* no se ejecutaban como se esperaba.

```
//Carroceria
view = camera.GetViewMatrix();
model = glm::mat4(1);
//model = glm::translate(model, PosIni + glm::vec3(movKitX, 0, movKitZ));
model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.6f, 0.6f, 0.6f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Carroseria.Draw(lightingShader);

//Llanta Delantera Der
view = camera.GetViewMatrix();
model = glm::mat4(1);
//model = glm::translate(model, PosIni + glm::vec3(movKitX, 0, movKitZ));
model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0));
//model = glm::translate(model, glm::vec3(1.7f, 0.5f, 2.6f));;
model = glm::scale(model, glm::vec3(0.6f, 0.6f, 0.6f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
LlantaDerSup.Draw(lightingShader);
```

```
//Movimiento circular del coche
if (circuito)
{
    if (recorrido1)
    {
        rotKit = 30;
        //slope(char varIndep, float posIniZ, float posFinZ, float posIniX, float posFinX)
        movKitZ = slope('x', 6.5, 25.5, 20.8, 9.9) * movX + 6.5;
        movX -= 0.1f;
        movKitX -= movX;
        printf("R1: movKitX: %f, movKitZ: %f \n", movKitX, movKitZ);
        if (movKitZ > 25.5)
        {
            printf("\n");
            /*recorrido1 = false;
            recorrido2 = true;*/
            circuito = false;
        }
    }
```

Como se aprecia, el coche debía tener su propio espacio de estados, donde cada estado indicaría la porción que el coche debía recorrer del circuito.