Laboratorio de Computación Grafica e Interacción Humano Computadora

Manual técnico Proyecto final: Alameda LEGO

García García Ricardo, Martínez Lopez Andrés
16 de mayo de 2020

Universidad Nacional Autónoma de México

Facultad de Ingeniería

Laboratorio de Computación Gráfica e Interacción Humano Computadora

Grupo 7

Semestre 2020-2

Equipo de trabajo:

Andrés Martínez López (Desarollador)

Nº de Cuenta: 314138567

Email: andres.mtz.cu@gmail.com

AndresMtzLpz

Ricardo García García (Desarollador)

Nº de Cuenta: 314256229

Email: rich030rich@gmail.com

PlugInRichi

Dirección del proyecto:

https://github.com/AndresMtzLpz/PROYECTO-LABGRAFICA

Objetivo

Crear un manual que permita identificar de manera práctica los puntos clave del desarrollo del proyecto y conocer su estructura para posibles modificaciones o actualizaciones.

Introducción

El proyecto requería el diseño de una alameda con estilo LEGO, dentro de las especificaciones se solicitaba la creación de ciertos elementos dentro de la alameda, como alumbrado, bancas, botes de basura, quisco y un personaje, de manera opcional se podían agregar elementos como iluminación para eventos y música de fondo.

El proyecto fue desarrollado en C++, haciendo uso de *Visual Studio* para la depuración y desarrollo de este, el proyecto se encuentra dividido en distintas clases que fueron proporcionadas por el profesor durante el semestre, adicionalmente, se añadieron clases extras con la finalidad de separar las funcionalidades del proyecto y mejorar la claridad del funcionamiento del programa.

A continuación de dará una breve explicación de lo que hace cada clase adicional así como los método que implementa y su utilidad en el programa.

Movimiento

Dentro de esta clase se realizan todas las operaciones necesarias para almacenar la posición, dirección y grado de giro para cada uno de los modelos, en general se especifica un función para cada modelo que requiera animación, estos métodos son llamados en la clase principal para indicar en la traslación de cada uno de los modelos después de haber llamado a la función *time()* que calcula un nuevo incremento en el tiempo y las animaciones

Clase: Movimiento			
Nombre del método	**Parámetros**	**Valor de retorno**	**Función**
movBici	`float movOffset, bool arranque_Bici`	`glm::vec3`	Retorna la posición de la bicicleta a través del tiempo
movBlackHawk	`float movOffset, bool despegue_BH`	`glm::vec3`	Retorna a posición del helicóptero de LEGO a través del tiempo

getGiroHelice		float	Retorna el grado de giro para el modelo del helicóptero de LEGO
giroBlackHawk		float	Retorna el giro que debe de dar el modelo del helicóptero de LEGO en función de su posición actual
giroBici		Float	Retorna el giro de la bicicleta en función de la posición actual.
giroSubeBaja		Float	Proporciona la animación para el sube y baja, regresa el giro del modelo.
getMovCir		glm::vec3	Retorna la dirección la luz de azul para eventos.
getMovLin		glm::vec3	Retorna la posición de la luz verde para eventos
getMovLin_2		glm::vec3	Retorna la posición de la luz amarilla para eventos
horaDia		Float	Retorna la hora del día utilizada para la activación de eventos que dependen de este valor
time		GLfloat	Realiza el cálculo del incremento para normalizar el tiempo en cualquier equipo.
getDeltaTime		float	Regresa el valor de DeltaTime calculado para ese instante de tiempo

Luminaria

Para implementar todas las luces necesarias en el proyecto se creó esta clase que crea todas las instancias necesarias para las luces, desde la clase principal se declaran los arreglos que serán enviados al shader, dichos arreglos se envían como parámetros al constructor de la clase para que almacene su dirección, a lo largo del programa se mandará a llamar al método *setLuminaria* cada que se necesite actualizar las luces que deben pintarse, en función de la hora del programa o el botón para encender las luces del quiosco. Adicionalmente, desde la clase principal se modifican las direcciones y posiciones de las luces cuando sea necesario (esto es útil para las luminarias móviles).

Clase: Luminaria			
Nombre del método	**Parámetros**	**Valor de retorno**	**Función**
getDirectional		`DirectionalLight*`	Proporciona la dirección de la luz tipo direccional a utilizar en función a la hora del día (almacenada como atributo de la clase).
getPointCount		`int`	Indica el número de instancias de este tipo de luminaria.
getSpotCount		`Int`	Indica el número de instancias de este tipo de luminaria.
setLuminaria	`int horaDia, bool botonK`		En función de la hora del día, y el estado del botón agrega o elimina instancias de luces de tipo SpotLight y PointLight de los arreglos que serán enviados al shader.
creaPointLights			Crea las instancias de todas las luces tipo pointlight que serán ocupadas.
creaSpotLights			Crea las instancias de todas las luces tipo spotlight que serán ocupadas.

Sonido

Esta clase sirve únicamente para proporcionar la música de fondo a todo el proyecto, haciendo uso de irrKlang se crean las instancias necesarias para reproducir un audio y detenerlo

Clase: Sonido			
Nombre del método	**Parámetros**	**Valor de retorno**	**Función**
`Reproduce`			Si el estado actual del audio está en reproducción no hace nada, de lo contrario crea el objeto necesario para reproducir una pista de audio.
`Deten`			Si el estado actual del audio es activo, detiene su reproducción y elimina los objetos relacionados a ella, en caso de que el estado sea inactivo no hace nada.

AnimacionKF

Contiene los métodos necesarios para almacenar las posiciones de los objetos a través del tiempo así como los métodos que permiten reconstruir la animación a haciendo uso de las posiciones almacenadas. El constructor de esta clase inicializa todas la variables necesarias para almacenar cada uno de los keyframes.

Clase: *AnimacionKF*			
Nombre del método	**Parámetros**	**Valor de retorno**	**Función**
`saveFrame`			Guarda la posición espacial del modelo para el que se están capturando los keyframes.
`resetElements`			Regresa el valor inicial a las posiciones del modelo que se esté animando con keyframes
`interpolation`			Realiza la interpolación de la posición entre dos valores almacenados de keyframes
`animate`			Es el método encargado de hacer las interpolaciones y calcular los incrementos para cada posición del modelo.
`animaAvion`			Contiene los valores iniciales para crear el arreglo de Keyframes que permitirá crear la animación.

movAvion		`glm::vec3`	Sirve vara recuperar la posición actual del modelo que se está animando, en este caso el avión.
rGiroAvionX		`Float`	Regresa el valor del giro del avión a través del eje X
rGiroAvionZ		`Float`	Regresa el valor del giro del avión a través del eje Z
rVueltaAvion		`Float`	Regresa el valor de el giro del avión a través del eje Y
inputKeyframes	`bool* keys`		Sirve para activar las opciones de entrada del la clase AnimacionKF, de esta manera se controla el número de acciones que se realizan al tocar una tecla.