Práctica Telas

ANIMACIÓN 3D

ROBERTO HERENCIAS MUÑOZ

Universidad
Rey Juan Carlos

Instrucciones

Ejecución de la simulación

Seleccionamos el “3D Object” Plane del apartado de Jerarquía de objetos de la interfaz de Unity, y en el Inspector vemos que hay un apartado para vincular un Script al objeto.

Añadimos el MassSpring.cs al apartado script del Inspector y saldrán todos los valores modificables public del script.

De manera similar, en el objeto Cube debemos añadir al apartado de script, el Fixer.cs. Esto hará que podamos asignar una “Tela” que es el objeto que será Fixeado al entrar en contacto con el objeto que contiene el script Fixer (cube); ponemos el objeto Plane en el apartado “Tela” y de esta forma el plano tendrá fixeados aquellos nodos que están en contacto con el objeto cube.

Finalmente, tras colocar los parámetros deseados, se da al play de la escena. (Se recomienda usar la cámara “Scene”, en lugar de la cámara “Game”, de esta forma podrá moverse en tiempo de ejecución el objeto “cube” para comprobar como la tela se mueve junto con el objeto Fixer).

Selección de parámetros

En la selección de parámetros del objeto plane, todos vienen por defecto colocados para una simulación óptima, pero puede cambiarse la masa de los nodos, el stiffness de los springs y la aceleración de la gravedad.

También puede cambiarse entre el método de integración “Symplectic” o “Explicit”.

Por último, se quita la pausa para que pueda dar comienzo la simulación, desmarcando el bool “Paused”.

Extras

En las implementaciones extra se ha incluido movimiento de la tela junto con el objeto fixer, como se ha comentado anteriormente, y además se programado amortiguamiento y viento.

Para controlar los parámetros de viento, tenemos en el Inspector un bool que activa y desactiva el viento en tiempo de ejecución. También se puede controlar la fricción del viento y cambiar la dirección en la que sopla este.

Y para el control de la cantidad de amortiguamiento, también tenemos un parámetro public en el Inspector, llamado damp.

En cuanto a los materiales, hay un material “Sujeción” y otro “Tela” simplemente decorativos que dan algo de color a la escena, y también hay un material “Tela_2sides” que contiene un shader para que el plano sea visible por ambas caras y de esta forma la tela se siga viendo cuando ondea, pero pierde la característica de profundidad y por ello no se ven sombras.

(El shader ha sido sacado de internet: https://forofenixdiscom.foroactivo.com/t61-problema-renderizado-de-tela-bandera)

Componentes implementados

Primer requisito

Para el primer requisito se ha creado un script MassSpring.cs donde se inicializan los valores public (para poder verlos desde Inspector) mass, stiffness, gravity, integrationMethod, etc. De esta forma, los valores mass de los nodos, y stiffness de los spring se definen a nivel de objeto. Además, se crea una lista de nodos y una lista de springs que serán utilizadas posteriormente.

En el método Start() [que posteriormente pasará a ser Awake()] se obtiene el mesh del objeto para obtener sus vértices y triángulos, y de esta forma crear los nodos y los springs.

Se obtiene la posición de cada vértice en posición global y se crea un nodo con esta posición y los valores de masa y gravedad inicializados anteriormente, que se añade a la lista de nodos.

- Los nodos son una clase (que no hereda de MonoBehaviour) con los atributos de posición, velocidad, fuerza, gavedad, masa y un bool “isFixed” que nos indica si ese nodo esta fixeado; y los métodos computeForces() que suma a los nodos la fuerza de la gravedad, y el método setFixed que se utiliza para poner el bool “isFixed” en true o false desde MassSpring.

Para la creación de springs se recorre el número total de triángulos y creando un spring para cada pareja de vértices, es decir, se crean 3 springs por cada triangulo, asignándoles al constructor los dos nodos que lo formas y además el stiffness inicializado anteriormente, y todo ello se añade a la lista de springs.

- Los spring son una clase (que no hereda de MonoBehaviour) con los atributos nodoA, nodoB, Length0, Length y stiffness; y los métodos updateLength() que guarda en Length la distancia entre la posición de nodoA y la posición de nodoB, y el método computeForces() que calcula la fuerza elástica con la longitud y el stiffness, además de añadir las fuerzas de los propios nodos A y B.

Finalmente, mediante el método Update de MassSpring se devuelven los vértices a coordenadas locales.

Los métodos stepExplicit() y stepSymplectic() calculan las fuerzas para cada instante de tiempo según el método de integración de Euler explícito y Euler simpléctico respectivamente.

Segundo requisito

Para el segundo requisito se ha creado un script Fixer.cs donde se declara un GameObject public donde deberemos asignar el componente que queremos que se quede fixeado al estar en contacto con el objeto con cualidad Fixer.

En este caso pondremos el script Fixer.cs al objeto cube, y le diremos que el GameObject que queremos que se vea afectado sea plane.

Como plane tiene asociado MassSpring.cs tiene nodos, y por tanto en Fixer.cs accedemos a los nodos del objeto asociado, para fixear aquellos nodos que estén en contacto con cube.

Extra

Como extra para este requisito se ha creado un valor posFixer, que guarda la posición en donde se encuentra el objeto cube, y en un método update(), va calculando si ha cambiado de posición para sumarle ese incremento de posición al valor pos de los nodos que están en la lista de fixeados, de esta forma la tela “plane” se mueve junto con el fixer “cube” en tiempo de ejecución.

Tercer requisito

Para evitar los springs repetidos y hacer springs de flexión que unan vértices opuestos de triángulos contiguos, antes de crear los springs de forma directa en MassSprings metemos todas las aristas en una lista Edges, de donde posteriormente iremos creando los springs.

- La lista Edge tiene como atributos los tres vértices de un triángulo, y los métodos de compare, que verifica si el vértice A y B de un triángulo son igual a los A y B de otro triángulo para de esta forma saber que son triángulos contiguos y por tanto crear una sola vez el spring AB y además hacer un spring de flexión de CC de cada uno de los triángulos; y un método CompareTo que viene heredado de IComparable<Edge> para poder usar .Sort() para ordenar la lista de Edges y que resulte más fácil comparar triángulos contiguos, pues su AB será igual.

En MassSpring crearemos el primer Spring con el primer AB de la lista de Edges, y posteriormente recorreremos la lista de Edges comparando el actual AB ([i]) con el anterior AB ([i-1]) creando un spring AB siempre que no sea igual que el del Edge anterior, y en caso de ser igual que el del anterior, no se crea el AB para no repetirse, pero si se crea un CC (spring de flexión).

Extras

Para crear amortiguamiento se crea un atributo damp que se le pasará a los spring junto con los nodos y el stiffness, y en la computeForces() del spring se le suma a la fuerza elástica la fuerza de amortiguamiento.

Para tener viento en la escena declaramos una velocidad para el mismo que al encontrarse en un vector, también nos indica la dirección de este, y una fricción del viento para calcular la fuerza de este sobre los nodos.

Además, mediante el bool “Wind” podemos activar o desactivar el viento en tiempo de ejecución.

Para crear el viento, durante la creación de los Edges, creamos también triángulos con la combinación de estos, y los nodos de esos triángulos son los que se verán afectados por el viento; para ello en el método de integración haremos una llamada al ComputeForces() de los triángulos justo después de haber llamado al ComputeForces() de los nodos.

- Los triángulos (formados cada uno de ellos por 3 nodos), reciben de MassSpring la fricción del viento y la velocidad de este, tras calcular el área del triángulo y obtener una velocidad que viene dada por la suma de las velocidades de cada nodo entre el número de nodos; calculamos la fuerza del viento y se reparte equitativamente entre cada nodo, añadiéndole está a la fuerza que estos nodos tenían previamente.