**Trabajo Práctico Nº1**

# Ejercicio 5 - Ejemplo de cuerpos blandos (SoftBody).

Analizar cómo se construye el SoftBody

En el archivo de código fuente **Game.cpp** se crean los cuerpos Softbody a partir de las siguientes líneas de código:

//instancia el objeto def del tipo struct b2ExSoftCircleBodyDef   
**b2ExSoftCircleBodyDef def;**

//define la cantidad de círculos que se vincularán  
**def.numParts = 10;**

//define el radio de los círculos  
**def.radius = 10.0f;**

//define el origen de los círculos, dónde aparecerán en la ventana  
**def.center = b2Vec2(0.0f, 15.0f);**

//define la blandura = resistencia de los círculos (todos igual), rango de valor de 0 a 1  
**def.softness = 0.5f;**

//instancia el objeto body (como puntero) del tipo struct b2ExSoftCircleBody y lo crea en el mundo con las definiciones guardadas en la variable def.  
**b2ExSoftCircleBody\* body = b2ExSoftCircleBody\_Create(world, &def);**

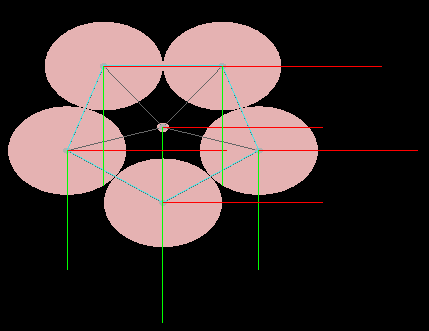
No presenta limitación en la cantidad de círculos (numParts), en el radio (radius), en la elección del origen (center), pero si se limita la blandura (softness) con la instrucción assert.

Explicar la forma en que se construye el SoftBody y Cómo afecta cada parámetro en su funcionamiento.

En los archivos b2ExSoftCircleBody.h encontramos dos construcciones:

* **struct b2ExSoftCircleBodyDef**; en esta se definen los datos del círculo blando como su centro, radio, cantidad de partes que están a su alrededor, rigidez, las propiedades de los sub círculos (los que están vinculados al cuerpo círculo blando) y finalmente las propiedades de los joint o articulaciones entre los cuerpos, como frecuencia y radio de amortiguación.
* El método **b2ExSoftCircleBody\_Create**, el que permite crear el, en el mundo, el cuero blando, según las configuraciones realizadas en Game.cpp (cantidad de sub círculos = numParts; radio de los sub círculos = radius; el centro del círculo blando = center; la rigidez de los cuerpos = softness = 0.85f; y la densidad de los cuerpos = density)

Ambos métodos son responsables de la construcción de un cuerpo blando (círculo blando) y sub cuerpos, todos dinámicos, vinculados por joint.



Analicemos cada construcción en particular:

**b2ExSoftCircleBodyDef** es una estructura (struct) definida y declarada en el archivo b2ExSoftCircleBody.h y que sirve como una colección de elementos o campos de distinto tipo para describir el circulo, entre ellos:

//método que define variables propias del círculo: centro origen en ventana, radio, número de partes, softness, densidad, fricción, la frecuencia y amortiguación de la articulación

b2ExSoftCircleBodyDef()

//variable vector2D center = origen del círculo en la ventana

b2Vec2 center;

//variable float radius = radio del círculo

float radius;

//variable int numParts = cantidad de círculos que se crearán alrededor del centro del círculo suave; cada parte es otro círculo vinculado mediante una articulación de distancia con los otros círculos y un cuerpo central circular

int numParts;

//variable float softness = blandura del círculo, variable que puede valer entre [0;1]

float softness;

//variable float desity = densidad a aplicar a la articulación

float density;

//variable float friction = fricción a aplicar a la articulación

float friction;

//variable float jointFrecuencyHz = frecuencia de la articulación

float jointFrequencyHz;

//variable float jointDampingRatio = amortiguación de la articulación

float jointDampingRatio;

**b2ExSoftCircleBodyf** es una estructura (struct) definida en el archivo b2ExSoftCircleBody.h y declarada en el archivo b2ExSoftCircleBody.cpp que sirve para coleccionar el número de partes alrededor del origen del cuerpo blando, y un cuerpo circular central, todos unidos mediante joint; las variables que están definidas en esta estructura son:

//variable int m\_numParts = numParts

int m\_numParts;

//matriz de punteros m\_parts de objetos Body = cantidad de círculos

b2Body\*\* m\_parts;

//puntero m\_center a objeto Body = origen donde se crea cada círculo

b2Body\* m\_center;

//matriz de punteros m\_partJoints de objetos Joint = cantidad de articulaciones

b2Joint\*\* m\_partJoints;

//matriz de punteros m\_centerJoints de objetos Joint = centros de las articulaciones

b2Joint\*\* m\_centerJoints;

//puntero m\_parts a objeto Word= mundo donde se aplicará la simulación

b2World\* m\_world;

Cuando se instancia el objeto body (como puntero) del tipo struct b2ExSoftCircleBody, este es creado por el método b2ExSoftCircleBody, hay que pasarle el cuerpo mundo (Word) y los datos que necesita la estructura b2ExSoftCircleBodyDef, coleccionados en el objeto def.

**b2ExSoftCircleBody\* body = b2ExSoftCircleBody\_Create(world, &def);**

Lo que hace este método es:

//recibir el dato del mundo (world y la colección de elementos de la estructura b2ExSoftCircleBodyDef)

b2ExSoftCircleBody\* b2ExSoftCircleBody\_Create(b2World\* world, const b2ExSoftCircleBodyDef\* def)

//declara una aserción para errores lógicos = que el softness elegido por el usuario esté dentro de los valores [0;1]

b2Assert(0.0f <= def->softness && def->softness <= 1.0f);

//instancia e inicia el puntero memory del tipo char = para asignar capacidad de memoria, con el tamaño de datos que necesita el círculo + los datos que necesita el cuerpo del centro + el tamaño de las articulaciones \* 2, según la cantidad de partes de que se definan en numParts.

char\* memory = (char\*) b2Alloc(sizeof(b2ExSoftCircleBody) + (sizeof(b2Body\*) + sizeof(b2Joint\*) \* 2) \* def->numParts);

//Si no hay asignación de memoria disponible se retorna Null

if (!memory)

return NULL;

//Instancia el puntero circle como objeto b2ExSoftCircleBody y lo inicia con la asignación de memoria, luego actualiza la variable memory con los valores que toma la variable circle

b2ExSoftCircleBody\* circle = (b2ExSoftCircleBody\*) memory;

memory += sizeof(b2ExSoftCircleBody);

circle->m\_world = world;

circle->m\_numParts = def->numParts;

circle->m\_parts = (b2Body\*\*) memory;

memory += sizeof(b2Body\*) \* def->numParts;

circle->m\_partJoints = (b2Joint\*\*) memory;

memory += sizeof(b2Joint\*) \* def->numParts;

circle->m\_centerJoints = (b2Joint\*\*) memory;

//variable constante del tipo float angleStep = el ángulo en radianes donde estará cada círculo desde el origen, dependiendo de la cantidad de círculos que se configure.

const float angleStep = (3.14159265358979323846f \* 2.0f) / def->numParts;

//variable constante del tipo float sinHalfAngle = calcula el seno del ángulo

const float sinHalfAngle = sinf(angleStep \* 0.5f);

//variable constante del tipo float subCircleRadius = calcula el radio de cada círculo externo

const float subCircleRadius = sinHalfAngle \* def->radius / (1.0f + sinHalfAngle);

//instancia el objeto sd como un CircleShape

b2CircleShape sd;

//define el radio del CircleShae = valor calculado en subCircleRadius

sd.m\_radius=subCircleRadius;

//por cada circle:

float angle = 0;

for (int i = 0; i < def->numParts; i++)

{

//define un vector 2D offset guarda los datos del radio de cada círculo menos el valor del radio del centro del origen con el centro de cada círculo.

b2Vec2 offset(sinf(angle), cosf(angle));

offset \*= def->radius - subCircleRadius;

//instancia un objeto bd del tipo b2Bodydef para definirlo como un cuerpo dinámico, la posición = centro (valor configurado en Game.cpp) del origen + el vector offset, luego crear en el mundo el objeto body del tipo b2Body, con las definiciones guardadas en bd, y asignarle al cuerpo body un fixture con el dato del radio m\_radius y la densidad (valor configurado en Game.cpp).

b2BodyDef bd;

bd.type= b2\_dynamicBody;

bd.position = def->center + offset;

b2Body\* body = world->CreateBody(&bd);

body->CreateFixture(&sd,def->density);

//cada elemento del arreglo de m\_parts, se le asigna el body creado antes, en una posición determinada

circle->m\_parts[i] = body;

//posición que cambiará al modificar el ángulo (del nuevo círculo) con un incremento calculado en angleStep.

angle += angleStep;

}

//crea el centro del origen donde se vincularán los círculos instanciando un objeto bd del tipo b2BodyDef, para definir su posición = centro (valor configurado en Game.cpp), un tipo de cuerpo blando, luego crear en el mundo el objeto body del tipo Circle, con las definiciones guardadas en bd, y asignarle un fixture con m\_raduis = al radius (valor configurado en Game.cpp) - valor calculado en subCircleRadius afectado por el valor de softness (valor configurado en Game.cpp), y la densidad de cada círculo externo (valor configurado en Game.cpp)

b2BodyDef bd;

bd.position = def->center;

bd.type=b2\_dynamicBody;

circle->m\_center = world->CreateBody(&bd);

sd.m\_radius = (def->radius - subCircleRadius \* 2.0f) \* (1.0f - def->softness);

circle->m\_center->CreateFixture(&sd,def->density);

//para crear los vínculos entre círculos instancia el objeto jointDef del tipo b2DistanceJointDef, y por cada parte (valor configurado en Game.cpp), se define una variable neighborIndex del tipo entero, que indica la cantidad de círculos externos.

b2DistanceJointDef jointDef;

for (int i = 0; i < def->numParts; i++)

{

const int neighborIndex = (i + 1) % def->numParts;

//inicializa los joint de cada arreglo de circle, elemento del arreglo indicado por el índice neighborIndex. Toma parámetros: body A = círcle [i], body B = circle[neighborIndex], y los vectores del origen y destino de acuerdo a los valores de i y neighborIndex, tomados desde el centro del mundo.

jointDef.Initialize(

circle->m\_parts[i], circle->m\_parts[neighborIndex],

circle->m\_parts[i]->GetWorldCenter(),

circle->m\_parts[neighborIndex]->GetWorldCenter());

//conecta cada joint

jointDef.collideConnected = true;

//define los valores de rigidez, amortiguación, frecuencia y radio de amortiguación definidos en la estructura b2ExSoftCircleBodyDef, para los cuerpos)

b2LinearStiffness(

jointDef.stiffness,

jointDef.damping,

def->jointFrequencyHz,

def->jointDampingRatio,

jointDef.bodyA,

jointDef.bodyB);

//se crea cada joint en m\_partJoint con los valores que tiene jointDef

circle->m\_partJoints[i] = world->CreateJoint(&jointDef);

// inicializa los joint de cada arreglo de circle, con el centro del cuerpo blando, tomando como parámetros: body A = círcle [i], body B = circle del cuerpo blando y los vectores del origen del círculo con el valor de center (valor configurado en Game.cpp)

jointDef.Initialize(

circle->m\_parts[i],

circle->m\_center,

circle->m\_parts[i]->GetWorldCenter(),

def->center);

//conecta cada joint

jointDef.collideConnected = true;

//define los valores de rigidez, amortiguación, frecuencia y radio de amortiguación definidos en la estructura b2ExSoftCircleBodyDef, para los cuerpos)

b2LinearStiffness(

jointDef.stiffness,

jointDef.damping,

def->jointFrequencyHz,

def->jointDampingRatio,

jointDef.bodyA,

jointDef.bodyB);

//se crea cada joint desde el centro del cuerpo blando en m\_centerJoint con los valores que tiene jointDef

circle->m\_centerJoints[i] = world->CreateJoint(&jointDef);

}

# Ejercicio 6 - ragdoll