Chipmunk para Bennu

20 de marzo de 2011

Orlando Aguilar Vivanco
Prg

Resumen

Lista y breve descripción de las funciones, estructuras, variables globales, locales y constantes de mod_chipmunk.

Índice

1. mod_chipmunk

Mod_Chipmunk es una librería, basada en Chipmunk, de física para BennuGD. Mod_Chipmunk lleva las funciones de la chipmunk original a BennuGD de una forma nativa, facilitando así el uso a cualquier usuario con tan solo setear algunas variables.

Chipmunk por su parte es una librería de física creada para C por Scott Lembcke. Su licencia dice así:

Mod_Chipmunk está basado en la versión 5.3.4 de la librería Chipmiunk, y en este documento describiré cada una de las variables globales, locales, estructuras y funciones que se pueden usar desde bennu para controlar la librería de física mod_chipmunk.

2. Globales

2.1. float gravity_X

Esta global es la gravedad en x, setearla cambiará la gravedad en el eje x.

2.2. float gravity_Y

Esta global es la gravedad en y, setearla cambiará la gravedad en el eje y.

2.3. float bias_coef

Su valor por defecto es 0.5, y se refiere a la forma en que se resuelve el solapamiento cada ejecución del motor. Un valor de 0.1 indica que después de 15 ejecuciones del motor el solapamiento se reducirá al 20 % y después de 30 se reducirá al 4 %. El máximo valor que se debiera usar es 1, sin embargo entre mayor sea el valor, más inestable se comportará la simulación frente a una gran cantidad de objetos apilados.

2.4. float collision_slop

Su valor por defecto es 0.1. Cuando se usa una escala distinta al pixel se puede variar este valor para corregir solapamientos y separaciones.

2.5. int contact_persistence

Su valor por defecto es 3 y se refiere a cuántas ejecuciones del motor se debe recordar dos objetos que colisionaban para la corrección de su posición.

2.6. int iterations

Se refiere a la cantidad de iteraciones que el motor debe realizar sobre cada proceso cada ejecución del motor. Su valor por defecto es 10.

2.7. float damping

Se refiere a la viscosidad del mundo y representa la forma en que la velocidad del objeto se va perdiendo. Su valor por defecto es 1.

2.8. float idleSpeedThreshold

Se refiere al límite de velocidad para que un objeto se considere inactivo. El valor 0 (por defecto) indica que se debe buscar este umbral automáticamente de acuerdo a la gravedad.

2.9. float sleepTimeThreshold

Su valor interfiere con la velocidad de la simulación permitiendo también ahorrar recursos en la computadora. Su valor por defecto es infinityf() y desactiva el algoritmo. El API oficial de la librería C dice así: "Time a group of bodies must remain idle in order to fall asleep".

2.10. float interval

Su valor por defecto es 1.0/25.0, y su valor más óptimo es 1/maxFPS. Se refiere al paso de tiempo que pasa en el motor físico en cada frame.

2.11. int phresolution

Se refiere a las veces que se debe ejecutar el motor cada frame. Su valor por defecto es 3. Mientras más grande sea el valor mejor será la simulación, pero es más costoso cada frame. Poner este valor a 0 permite pausar el motor físico.

2.12. cpVect cpvzero

Este vector especial es el vector (0,0). Existe sólo para operaciones que requieran este vector. Su valor no se debe modificar.

3. Estructuras

3.1. cpVect

```
TYPE cpVect
float x; float y;
END
```

Esta estructura es un vector.

3.2. cpBB

```
TYPE cpBB
float l, b, r ,t;
END
```

Esta estructura almacena un boundingbox que posteriormente se puede usar en algunas funciones.

3.3. cpsegmentqueryinfo

```
TYPE cpsegmentqueryinfo
int shape,id;
float x,y,t;
cpVect n;
END
```

En esta estructura se almacena el resultado de un raycol. Sus elementos son el shape con el que colisionó, el punto en el que colisionó, la distancia entre el comienzo el raycol y el punto (x,y) en el rango (0,1), el vector normal de la superficie con la que colisionó y el id del proceso con el que colisionó el raycol, o cero si el shape no tiene proceso (shapes del scroll o del background).

3.4. tPoints

```
TYPE tPoints
cpVect point, normal;
float dist;
end
```

Estas estructuras forman parte de la estructura cpContactPointSet. Points tiene las coordenadas (x,y) donde dos procesos han colisionado y la normal de la colisión. Dist se refiere a la distancia de solapamiento que se ha corregido.

3.5. cpContactPointSet

```
TYPE cpContactPointSet
int count;
int id1, id2;
int shape1,shape2;
tPoints points[6];
end
```

Esta estructura es usada en el handler de historial de colisiones. En ella se almacenan los puntos en que los procesos id1 e id2 colisionan. id1 e id2 son el id de los procesos que cumplen con los requisitos del handler y que han colisionado y shape1 y shape2 son los shapes que colisionaron de estos procesos.

4. Constantes

4.1. Constantes de shapes

4.1.1. CP_NO_GROUP

Esta constante indica que el shape en cuestión no pertenece a ningún grupo

4.1.2. CP_ALL_LAYERS

Indica que el shape en custión se encuentra en todas las capas.

4.1.3. CP_C_GROUP

Entero con el grupo al que pertenece un shape.

4.1.4. CP_C_LAYERS

Entero con las capas a la que pertenece un shape. Las capas se usan en funciones de colisiones. Dos procesoso colisionan si la expresión (a.layers & b.layers) != 0 se cumple.

4.2. Formas del Shape

4.2.1. TYPE_NONE

Esta constante indica que el proceso no se encuentra dentro del motor físico.

4.2.2. TYPE_EMPTY

Esta constante indica un body vacío (sin shape).

4.2.3. TYPE_BOX

Esta forma del shape es un cuadro con cuatro vértices, cada uno en cada esquina del gráfico del proceso.

4.2.4. TYPE_CONVEX_POLYGON

Esta forma indica un polígono convexo con cualquier número de vértices para representar cualquier forma convexa.

4.2.5. TYPE_LINE

Esta forma es una linea formada por dos puntos extremos y un ancho de línea.

4.2.6. TYPE_CIRCLE

Esta forma es circular.

4.3. Constantes referentes a propiedades del Body

4.3.1. CP_C_V

Vector que define la velocidad de un proceso (el body del proceso).

4.3.2. CP_C_F

Vector que define la fuerza que se aplica a un proceso (el body del proceso).

4.3.3. CP_C_W

Flotante con la velocidad de rotación actual del proceso.

4.3.4. CP_C_V_LIMIT

Flotante con la máxima velocidad que un cuerpo (proceso) puede tener antes de actualizar su velocidad.

4.3.5. CP_C_W_LIMIT

Flotante con la máxima velocidad de rotación que un cuerpo (proceso) puede tener antes de actualizar su velocidad de rotación.

4.3.6. CP_C_T

Flotante con el torque aplicado al proceso.

4.3.7. CP_C_ROT

Vector actual de rotación de un body.

4.3.8. CP_C_SURFACE_V

Velocidad de la superficie del objeto. De tipo cpVect. Útil para crear cintas transportadoras.

4.3.9. CP_C_E

Esta propiedad es la elasticidad de un shape. Su tipo es float.

4.3.10. CP_C_U

Esta propiedad es la fricción de un shape. Su tipo es float.

4.4. Constantes de las propiedades de constraints

4.4.1. CP_C_CA

Body A de un constraint.

4.4.2. CP_C_CB

Body B de un constraint.

4.4.3. CP_C_MAXFORCE

Es la máxima fuerza que un constraint puede usar para actuar sobre los cuerpos. El valor por defecto es infinity.

4.4.4. CP_C_BIASCOEF

Se refiere a la corrección de la posición de los cuerpos cada frame*phresolution.

4.4.5. CP_C_MAXBIAS

Es la máxima velocidad que un constraint puede usar para actuar sobre los cuerpos. El valor por defecto es infinity.

4.4.6. CP_C_ANCHR1

Se refiere al punto donde el constraint se sujetará del primer proceso. Las coordenadas son locales.

4.4.7. CP_C_ANCHR2

Se refiere al punto donde el constraint se sujetará del segundo proceso. Las coordenadas son locales.

4.4.8. CP_C_DIST

Se refiere a la distancia que los Constraints deben mantener.

4.4.9. CP_C_MIN

Se refiere a la mínima distancia (o ángulo en grados) que el Constraint tiene permitido mantener.

4.4.10. CP_C_MAX

Se refiere a la máxima distancia (o ángulo en grados) que el Constraint tiene permitido mantener.

4.4.11. CP_C_GROOVEB

Se refiere al final de la línea que un Constraint de tipo Groove recorre (x,y).

4.4.12. CP_C_GROOVEA

Se refiere al principio de la línea que un Constraint de tipo Groove recorre (x,y).

4.4.13. CP_C_RESTLENGTH

Se refiere a la distancia que un resorte debe mantener

4.4.14. CP_C_STIFFNESS

Se refiere a la rigidez que un resorte tiene.

4.4.15. CP_C_DAMPING

Se refiere a la amortiguación de un resorte.

4.4.16. CP_C_RESTANGLE

Se refire al ángulo relativo (en grados) que dos cuerpos deben mantener.

4.4.17. CP_C_ANGLE

Se refire al ángulo de un RatchetJoint.

4.4.18. CP_C_PHASE

Se refiere a la fase de un ratchetJoint.

4.4.19. CP_C_RATCHET

Se refiere al Ratchet de un RatchetJoint.

4.4.20. CP_C_RATIO

Se refiere a la relación de la velocidad angular entre dos cuerpos.

4.4.21. CP_C_RATE

Se refiere a la velocidad angular deseada de un motor.

5. Locales

5.1. int body

Esta local almacena el id del body del proceso. Se puede usar en funciones que requieran este id. Nunca debe modificarse su valor ya que puede desestabilizar el programa.

5.2. int shape

Esta local almacena el id del shape principal del proceso. Un proceso puede tener todos los shapes que se requieran para darle el efecto de colisiones deseado, pero sólo uno de ellos (el primero en crearse) se almacenará en esta local. Para poder acceder a los demás shapes existe una función INT GETSHAPES(INTEGER) que retorna los shapes que tiene un proceso o 0 si ya ha retornado todos. Esta local se puede usar para las funciones que lo requieran, pero su valor nunca debe ser modificado manualmente.

5.3. int incr_x

Esta local indica los pixeles que un procoso se debe mover, en el eje x, el próximo frame, a partir de su posición actual. Su valor se setea a 0 cada frame.

5.4. int incr_y

Esta local indica los pixeles que un procoso se debe mover, en el eje y, el próximo frame, a partir de su posición actual. Su valor se setea a 0 cada frame.

5.5. float inertia

Esta local es el momento de inercia del proceso.

5.6. float mass

Esta local es la masa del proceso.

5.7. int static

Su valor puede ser true o false, e indica si el proceso es estático o no. Su valor es considerado sólo a la hora de crear el cuerpo, y cada vez que el proceso haga un nuevo cuerpo, pero no después.

5.8. float elasticity

Esta variable es la elasticidad del shape principal de un proceso. Si un proceso tiene más de un shape, se tiene que seleccionar manualmente la elasticidad de los demás shapes mediante la función INT DEFSHAPEF(INTEGER, INTEGER, FLOAT). Este valor indica cómo debe reaccionar el cuerpo frente a otros, si por ejemplo dos cuerpos colisionan y la elasticidad de ambos es mayor que cero, se multiplicará la elasticidad de ambos y el valor indicará qué tanto rebote se generará entre ambos.

5.9. float friction

Esta variable es la fricción del shape principal de un proceso. Si un proceso tiene más de un shape, se tiene que seleccionar manualmente la fricción de los demás shapes mediante la función INT DEFSHAPEF(INTEGER, INTEGER, FLOAT). Si la fricción de un objeto es 0, no rotará al friccionar con otros.

5.10. int group

Su valor por defecto es CP_NO_GROUP, que indica que el shape principal no pertenece a ningún grupo. Si un proceso tiene más de un shape, se tiene que seleccionar manualmente el grupo de los demás shapes mediante la función INT DEFSHAPEI(INTEGER, INTEGER, INTEGER).

Shapes del mismo grupo no colisonarán aunque pertenezcan a distintos procesos. Es importante señalar que los shapes del mismo proceso nunca presentarán colisión entre sí.

5.11. int layers

Su valor por defecto es CP_ALL_LAYERS que quiere decir que pertenecen a todas las capas. sólo shapes en la misma capa reportan colisión.

Las colisiones en las capas no es una simple comparación de valores, en realidad es un and entre las capas (a.layer & b.layer)!=0. Si aplicar un and bit a bit entre el valor de las capas retorna siempre cero, los shapes están en distinta capa; si alguno retorna uno, se considera que ambos shapes están en las misma capa. Si un proceso tiene más de un shape, se tiene que seleccionar manualmente la capa de los demás shapes mediante la función INT DEFSHAPEI(INTEGER, INTEGER, INTEGER).

5.12. int ShapeType

Su valor por defecto es TYPE_NONE que le indica al motor que el proceso no pertenece a la física del juego. Los posibles valores que la variable puede tomar y una descripción de ellos aparece en 4.2.

5.13. int * params

Su valor por defecto es null, este punter se usa para indicarle al creador de cuerpos físicos con qué parámetros queremos crear el cuerpo. Normalmente el creador de cuerpos creará el cuerpo de forma automática, pero si este puntero es distinto de null, se usarán los valores contenidos aquí para crear el cuerpo.

El arreglo que se pase a este parámetro debe tener la siguiente forma:

Si el ShapeType es igual a TYPE_LINE, se pondrá en el primer campo del arreglo la coordenada x del punto en donde comienza la línea, luego la coordenada y, luego la coordenada x del punto donde termina, en seguida la coordenada y y por último el radio de la línea. Todas las coordenadas son locales, y el centro del proceso es el punto (0,0).

Si el ShapeType es TYPE_CIRCLE, el primer campo del arreglo debe ser la coordenada x del centro del círculo, el segundo es la coordenada y, y el tercero es el radio del círculo. Todas las coordenadas son locales, y el centro del proceso es el punto (0,0).

Si el ShapeType es TYPE_CONVEX_POLYGON o TYPE_BOX, el primer elemento es la coordenada x del centro del polígono, el segundo es la coordenada y, en seguida tenemos la cantidad de vértices y luego la lista de los vértices (x,y). El orden de los vértices es el contrario al de las manecillas del relog.

5.14. int collisionType

Su valor por defecto es 0 y su uso es en los handlers de collisión.

En el handler se pide que se reporte el historial de colisión de los objetos cuyo collisionType es igual a algún valor que el programador ha prestablecido al crear el handler.

6. Funciones de mod_chipmunk

6.1. Funciones matemáticas

6.1.1. INFINITYF()

Retorna el número infínito para los flotantes. Útil en las funciones que aceptan este valor.

6.1.2. FLOAT DEG2RAD(FLOAT)

Cambia un ángulo de grados a radianes.

6.1.3. FLOAT CPFLERP(FLOAT a, FLOAT b, FLOAT t)

Retorna la interpolación lineal entre a y b.

6.1.4. FLOAT CPFCLAMP(FLOAT f, FLOAT min, FLOAT max)

Hace que el valor de f esté entre min y max.

6.1.5. FLOAT CPFLERPCONST(FLOAT f1, FLOAT f2, FLOAT d)

Realiza la interpolación lineal de f1 a f2 en no más de d.

6.1.6. INT CPVEQL(POINTER, POINTER)

verifica si dos vectores soniguales.

6.1.7. INT CPVADD(POINTER a, POINTER b, POINTER res)

suma dos vectores y rellena el vector res con el resultado.

6.1.8. INT CPV(POINTER a, POINTER b, POINTER res)

Resta dos vectores y rellena el vector res con el resultado

6.1.9. INT CPVNEG(POINTER a, POINTER res)

Niega el vector a y rellena res con el resultado.

6.1.10. INT CPVMULT(POINTER a, FLOAT b, POINTER res)

Multiplica el vector a por b y rellena el vector res con el resultado.

6.1.11. INT CPVPERP(POINTER a, POINTER res)

Rellena el vector res con el vector perpendicular a 'a' (90 grados).

6.1.12. INT CPVRPERP(POINTER a , POINTER res)

Rellena el vector res con el vector perpendicular a 'a' (-90 grados).

6.1.13. INT CPVNORMALIZE(POINTER a , POINTER res)

Rellena el vector res con una copia normalizada del vector normal a.

6.1.14. INT CPVNORMALIZE_SAFE(POINTER a , POINTER res)

Rellena el vector res con una copia normalizada del vector normal a. Si el vector a es (0,0) retorna (0,0) protegiéndose de divisiones entre 0.

6.1.15. INT CPVPROJECT(POINTER a, FLOAT b, POINTER res)

Rellena el vector res con la proyección de a sobre b.

6.1.16. INT CPVROTATE(POINTER a, FLOAT b, POINTER res)

Rota el vector a por b y rellena res con el resultado.

6.1.17. INT CPVUNROTATE(POINTER a, FLOAT b, POINTER res)

Función inversa a CPVROTATE.

6.1.18. INT CPVFORANGLE(FLOAT a, POINTER res)

Rellena res con un vector unitario que tenga el ángulo a en radianes.

6.1.19. INT CPVCLAMP(POINTER a, FLOAT b, POINTER res)

Restringe el vector "a" a la longitud "b" y rellena el vector "res" con el resultado.

6.1.20. INT CPVLERP(POINTER v1, POINTER v2, FLOAT t, POINTER res)

Rellena el vector res con la interpolación lineal de v1 y v2.

6.1.21. INT CPVLERPCONST(POINTER v1, POINTER v2, FLOAT d, POINTER res)

Rellena el vector res con la interpolación lineal v1 y v2 con distancia igual a d.

6.1.22. INT CPVSLERP(POINTER v1, POINTER v2, FLOAT t, POINTER res)

Rellena el vector res con la interpolación esférica lineal de v1 y v2.

INT CPVSLERPCONST(POINTER v1, POINTER v2, FLOAT a, POINTER res) Rellena el vector res con la interpolación esférica lineal de v1 y v2 con un ángulo máximo de a en radianes.

6.1.23. INT CPVNEAR(POINTER a, POINTER b, FLOAT c)

Retorna true si la distancia entre a y b es menor que c.

6.1.24. FLOAT CPVDOT(POINTER a, POINTER b)

Retorna el producto punto entre a y b.

6.1.25. FLOAT CPVCROSS(POINTER a, POINTER b)

Retorna la magnitud de z del producto cruz de los vectores a y b.

6.1.26. FLOAT CPVLENGTH(POINTER a)

Retorna la longitud del vector a.

6.1.27. FLOAT CPVLENGTHSQ(POINTER a)

Retorna la longitud al cuadrado del vector a, siendo más rápido este cálculo cuando sólo se quieren comparar magnitudes.

6.1.28. FLOAT CPVTOANGLE(POINTER a)

Retorna el ángulo del vector a en radianes.

6.1.29. FLOAT CPVDISTSQ(POINTER a, POINTER b)

Retorna la distancia al cuadrado entre los vectores a y b.

6.1.30. FLOAT CPVDIST(POINTER a, POINTER b)

Retorna la distancia entre los vectores a y b.

6.2. Funciones de colisiones y handlers

6.2.1. FLOAT SEGMENTQUERYHITDIST(INTEGER, INTEGER, INTEGER, INTEGER, POINTER)

Retorna la longitud de un raycol desde su comienzo hasta el punto de colisión. Los parámetros son el comienzo del raycol en x y en y, el final en x y en "y" y la estructura rellenada por la función SpaceSegmentQueryFirst.

6.2.2. INT SEGMENTQUERYHITPOINT(INTEGER, INTEGER, INTEGER, INTEGER, POINTER, POINTER, POINTER)

Permite conocer el punto donde un raycol ha colisionado. Los parámetros son el comienzo del raycol en (x,y), el final en (x,y), la estructura rellenada por la función SpaceSegmentQueryFirst y dos punteros a enteros para rellenarlos con este punto.

6.2.3. INT CPSHAPESEGMENTQUERY(INTEGER shape, POINTER p1, POINTER p2, POINTER cpSegmentQueryInfo)

Verifica si el shape es tocado por la linea que va de p1 a p2 y rellena la estructura cpSegmentQueryInfo con la información de la colisión.

6.2.4. INT CPSEGMENTQUERYHITPOINT(POINTER p1, POINTER p2, POINTER cpSegmentQueryInfo, POINTER res)

Rellena res con el punto de colisión de un raycol. Los parámetros son el punto de incio del raycol y el punto final además de un puntero a una estructura de tipo cpSegmentQueryInfo que previamente debió haber sido rellenada con la función CPSPACESEGMENTQUERYFIRST o alguna otra.

6.2.5. FLOAT CPSEGMENTQUERYHITDIST(POINTER p1, POINTER p2, POINTER cpSegmentQueryInfo)

Retorna la distancia entre el inicio de un raycol y el punto de colisión. CpSegmentQueryInfo debió haber sido rellenada previamente con alguna función como CPSPACESEGMENTQUERYFIRST o alguna otra.

6.2.6. INT CPSPACESEGMENTQUERYFIRST(POINTER p1, POINTER p2, INTEGER layer, INTEGER group, POINTERcpSegmentQueryInfo)

Retorna el Id del primer proceso que toca la linea que comienza en p1 y termina en p2. También rellena la estructura cpSegmentQueryInfo. Layer y group se refiere a las capas y el grupo de los shapes que puede usar la función.

6.2.7. INT CPSPACEPOINTQUERYFIRST(POINTER p, INTEGER layer, INTEGER group)

Retorna el shape que toque el punto p. Los parámetros son el cpVect p, las capas dónde buscar y el grupo del shape que se busca.

6.2.8. INT CPSHAPEPOINTQUERY(INTEGER shape, POINTER p)

Verifica si el shape toca el punto p de tipo cpVect.

6.2.9. INT CPSPACERESIZESTATICHASH(INTEGER space, FLOAT dim, INTEGER count), INT CPSPACERESIZEACTIVEHASH(INTEGER space, FLOAT dim, INTEGER count)

La información de estas funciones la colocaré tal cual viene en la ayuda de Chipmunk C.

The spatial hash data structures used by Chipmunk´s collision detection are fairly size sensitive. dim is the size of the hash cells. Setting dim to the average collision shape size is likely to give the best performance. Setting dim too small will cause the shape to be inserted into many cells, setting it too low will cause too many objects into the same hash slot.

count is the suggested minimum number of cells in the hash table. If there are too few cells, the spatial hash will return many false positives. Too many cells will be hard on the cache and waste memory. the Setting count to ~10x the number of objects in the space is probably a good starting point. Tune from there if necessary. By default, dim is 100.0, and count is 1000. The new demo program has a visualizer for the static hash. You can use this to get a feel for how to size things up against the spatial hash.

6.2.10. INT SPACEPOINTQUERYFIRST(FLOAT x, FLOAT y, INTEGER layer, INTEGER group)

Retorna el prceso que toca el punto (x,y) filtrando por layer y group (el filtro se hace mediante los campos layer y group de los shapes del proceso).

6.2.11. INT SPACESEGMENTQUERYFIRST(FLOAT, FLOAT, FLOAT, FLOAT, INTEGER, INTEGER, POINTER cpSegmentQueryInfo)

Retorna el proceso que toca el raycol creado desde (x1,y1) hasta (x2,y2) y filtrado por layer y group. También rellena cpSegmentQueryInfo con la información de la colisión para uso en funciones como SegmentQueryHitPoint o SegmentQueryHitDist.

6.2.12. INT COLLISIONHANDLERNEW(INTEGER collisionType1, INTEGER collisionType2)

Crea un handler que almacena las colisiones de shapes con collisionType igual a collisionType1 y collisionType2. El handler se elimina automáticamente al hacer cleanSpace o al cerrar el programa.

Esta función retorna el identificador del handler para usos posteriores.

6.2.13. INT REMOVECOLLISIONHANDLER(INTEGER handler)

Remueve manualmente el handler.

6.2.14. INT GETCOLLISIONINFO(INTEGER handler, POINTER cpContactPointSet)

Accede al historial de este frame del handler almacenando la información en el puntero cpContact-PointSet.

La función retorna 1 si hay información de historial colisión, de lo contrario retorna 0.

6.3. Funciones de Bounding Box

6.3.1. INT CPBBWRAPVECT(POINTER bb, POINTER v, cpVect POINTER res)

Rellena res con una copia de v envuelto en el boundingBox bb.

6.3.2. INT CPBBCLAMPVECT(POINTER bb, POINTER v, cpVect POINTER res)

Rellena res con una copia de v restringido al boundingBox bb.

6.3.3. INT CPBBEXPAND(POINTER bb, POINTER v, cpBB POINTER bbres)

Rellena a bbres con el boundingbox mínimo que contiene al boundingbox bb y al vector v.

6.3.4. INT CPBBCONTAINSVECT(POINTER bb, POINTER v)

Retorna true si el boundingbox bb contiene al vector v.

6.3.5. INT CPBBMERGE(POINTER bba , POINTER bbb , POINTER res)

Rellena el boundingbox res con el mínimo boundingbox que continee a bba y bbb.

6.3.6. INT CPBBCONTAINSBB(POINTER bba, POINTER bbb)

Retorna true si el boundingbox bba contiene al boundingbox bbb.

6.3.7. INT CPBBINTERSECTS(POINTER bba, POINTER bbb)

Retorna true si bba intersecta con bbb.

6.3.8. INT CPBBNEW(FLOAT l, FLOAT b, FLOAT r, FLOAT t, POINTER res)

Rellena el boundingbox res con los valores (l,b,r,t) para construirlo.

6.3.9. INT INTERSECTS(INTEGER id1, INTEGER id2)

Retorna true si los procesos id1 e id2 intersectan en sus boundingbox.

6.3.10. INT CONTAINS(INTEGER id1, INTEGER id2)

Retorna true si el boundingbox de id1 contiene completamente al boundingbox de id2.

6.3.11. INT CONTAINSVEC(INTEGER id, FLOAT x, FLOAT y)

Retorna true si el boundingbox de id contiene el punto (x,y).

6.4. Funciones de procesos

Funciones del body

6.4.1. INT SHAPECACHEBB(INTEGER id, POINTER bb)

Almacena en bb el boundingbox del proceso id.

6.4.2. INT FORCECREATEBODY()

El cuerpo físico de un proceso se crea después del primer frame, sin embargo si se requiere el cuerpo antes del frame se puede usar esta función la cual crea el cuerpo físico inmediatamente. También permite cambiar el tipo de cuerpo físico de un proceso o incluso eliminarlo.

Para eliminar el cuerpo físico se debe usar un shapeType igual a type_none y el cuerpo se eliminará el próximo frame, pero si se usa esta función se eliminará inmediantamente.

6.4.3. INT DEFBODYF(INTEGER, INTEGER, FLOAT)

Permite definir propiedades de un body con flotantes. Los parámetros son el body (local de todos los procesos), el campo a modificar el body y el nuevo valor.

Los campos que podemos definir son:

- CP_C_V_LIMIT
- CP_C_W_LIMIT

6.4.4. INT DEFBODYI(INTEGER, INTEGER, INTEGER)

No usado actualmente

6.4.5. FLOAT GETBODY(INTEGER body, INTEGER a)

Retorna un campo de un body. Los parámetros son el body (local de todos los procesos), y el elemento que se quiere conocer. El valor retornado es un flotante. Los valores que se pueden conocer son:

- CP_C_W
- CP_C_T
- CP_C_V_LIMIT
- CP_C_W_LIMIT

6.4.6. INT GETBODY(INTEGER body, INTEGER a, POINTER res)

Rellena el puntero res con el valor del body que se solicita con a.

Los valores que se pueden conocer son:

- CP_C_V
- CP_C_F
- CP_C_ROT

6.4.7. INT DEFBODYP(INTEGER, INTEGER, POINTER)

Permite definir propiedades de un body con punteros. Los parámetros son el body (local de todos los procesos), el campo a modificar el body y el puntero con la estructura con los nuevos valores.

Los campos que podemos definir son:

- CP_C_V

Si se usa esta función y se define CP_C_V, se debe usar la función UpdateVelocity() o alguna de sus variantes.

6.4.8. FLOAT CPMOMENTFORCIRCLE(FLOAT m, FLOAT v1, FLOAT v2, POINTER v)

Retorna el momento de inercia de un círculo cuya masa es m, su centro de gravedad es v, su diámetro interno es v1 y su diámetro externo es v2. Si el círculo no está hueco su diámetro interno debe ser 0.

6.4.9. FLOAT CPMOMENTFORSEGMENT(FLOAT m, POINTER a, POINTER b)

Retorna el momento de inercia de una linea cuya masa es m y sus puntos de comienzo y final son los puntos definidos por los vectores a y b. A y b deben estar en coordenadas locales al proceso.

6.4.10. FLOAT CPMOMENTFORPOLY(FLOAT m, INTEGER n, POINTER l, POINTER c)

Retorna el momento de inercia de un polígono convexo cuya masa es m, el número de vértices es n, la lista de vértices es l y su centro de gravedad es el vector c. Las listas de vértices se forman de vectores de cpvect.

6.4.11. FLOAT CPMOMENTFORBOX(FLOAT m, FLOAT w, FLOAT h)

Retorna el momento de inercia del rectángulo cuya masa es m, y tiene ancho w y alto h.

6.4.12. GETOPTIMALINERTIA(INTEGER type_shape,INTEGER shape)

Retorna la inercia para un shape de tipo círculo o línea.

6.4.13. GETOPTIMALINERTIA(INTEGER type_shape,INTEGER shape,FLOAT x0,FLOAT y0)

Retorna la inercia para un shape de tipo TYPE_CONVEX_POLYGON o TYPE_BOX. (x0,y0) es el centro del polígono.

6.4.14. FLOAT CPAREAFORCIRCLE(FLOAT a, FLOAT b)

Retorna el área del círculo cuyo diámetro interno es a y cuyo diámetro externo es b.

6.4.15. FLOAT CPAREAFORSEGMENT(POINTER a, POINTER b, FLOAT d)

Retorna el área de la linea cuyos puntos de inicio y fin (en coordenadas locales) son a y b y cuyo diámetro es d.

6.4.16. FLOAT CPAREAFORPOLY(INTEGER n, POINTER l)

Retorna el área del polígono convexo formado por la lista de vértices "l". Esta lista de vértices debe tener longitud n (número de vértices). Las listas de vértices se forman con un arreglo de vectores de cpVect.

6.4.17. INT SLEW(FLOAT x, FLOAT y, FLOAT dt)

Mueve el actual proceso a los puntos (x,y) usando como factor de velocidad dt (dt suele ser interval/phresolution). Usar esta función requiere llamar UPDATEVELOCITY hasta que el proceso deje de moverse

6.4.18. INT UPDATEVELOCITY()

Actualiza la velocidad de un proceso disminuyéndola de acuerdo a las propiedades del mundo.

6.4.19. INT UPDATEVELOCITY(FLOAT damping)

Actualiza la velocidad de un proceso disminuyéndola de acuerdo damping.
Almacena en x2 y y2 las coordenadas locales obtenidas de x1 e y1 usando el id de algún proceso.

6.4.20. INT APPLYIMPULSE(INTEGER id, FLOAT x1, FLOAT y1, FLOAT x2, FLOAT y2)

Aplica el impulso (x1,y1) al proceso id en el punto (x2,y2).

6.4.21. INT RESETFORCES(INTEGER id)

Establece las fuerzas de id a cero.

6.4.22. INT APPLYFORCE(INTEGER id, FLOAT x1, FLOAT y1, FLOAT x2, FLOAT y2)

Aplica el impulso (x1,y1) al proceso id en el punto (x2,y2).

6.4.23. INT SLEEP(INTEGER)

Actualmente un proceso se duerme usando signal(id,s_sleep ó s_freeze), por lo tanto esta función no tiene ningún efecto.

6.4.24. INT ISSTATIC(INTEGER id)

Retorna true si id es estático.

6.4.25. INT ISROGUE(INTEGER id)

Retorna true si id está en el mundo (no es estático).

6.4.26. INT ISSLEEPING(INTEGER id)

Retorna true si id proceso está durmiendo.

6.4.27. INT ACTIVATEPROCESSTOUCHINGME()

Actualmente no funciona. La forma de dormir y despertar cuerpos en la física es mediante signal de bennu.

Funciones del shape

6.4.28. INT ADDCIRCLESHAPE(FLOAT x, FLOAT y, FLOAT r)

Permite agregar otro shape de tipo circle a un proceso con radio r y centro (x,y).

Esta función retorna el indetificador del shape para usos posteriores con otras funciones como defShapeF.

6.4.29. INT ADDSEGMENTSHAPE(FLOAT x1, FLOAT y1, FLOAT x2, FLOAT y2, FLOAT r)

Permite agregar otro shape de tipo Línea con inicio (x1,y1), fin (x2,y2) y radio r.

Esta función retorna el indetificador del shape para usos posteriores con otras funciones como defShapeF.

6.4.30. INT ADDPOLYSHAPE(FLOAT x, FLOAT y, INTEGER n, POINTER l)

Permite agragar otro shape de tipo polígono convexo a un proceso con centro (x,y), cantidad de vértices n y lista de vértices l (array de cpVect).

Esta función retorna el indetificador del shape para usos posteriores con otras funciones como defShapeF.

6.4.31. INT GETSHAPES(INTEGER)

Retorna un shape del proceso cada vez que sea llamada la función hasta que no haya más constraints retornando cero.

La función se reinicia automáticamente cada frame.

Con esta función se puede recorrer la lista de shapes para buscar alguno en específico o para hacer algo con todos ellos.

6.4.32. INT ADDINANIMATESHAPE(INTEGER, FLOAT, FLOAT, FLOAT)

6.4.33. INT ADDINANIMATESHAPE(INTEGER, FLOAT, FLOAT, FLOAT, FLOAT, FLOAT)

6.4.34. INT ADDINANIMATESHAPE(INTEGER, FLOAT, FLOAT, INTEGER, POINTER)

Agrega un shape (Cuerpo de colisión) al fondo o scroll sin necesidad de que haya un proceso que lo mantenga vivo. Estos shapes son eliminados con cleanSpace().

Los parámetros son el tipo de shape:

TYPE_BOX, TYPE_CONVEX_POLYGON, TYPE_LINE, TYPE_CIRCLE.

Los demás parámetros dependen del tipo:

En el caso de que el tipo sea TYPE_CIRCLE, el siguiente parámetro es el x del centro del círculo, el y del centro del círculo y el radio del círculo.

Si el tipo es Line los siguientes parámetros son el x inicial de la linea, el y inicial, el x final, el y inicial y el radio de la línea.

Si el tipo es TYPE_CONVEX_POLYGON o TYPE_BOX el siguiente parámetro es el x del centro del polígono, el y del centro, el número de vértices y la lista de vértices (array de cpVect).

INT DEFSHAPEI(INTEGER, INTEGER, INTEGER) Define la propiedad de un shape usando un entero. Las propiedades de un shape que se pueden definir son:

- CP_C_GROUP
- CP_C_LAYERS

INT GETCPSHAPEI(INTEGER shape, INTEGER campo) Retorna los valores del shape que son enteros. Campo puede ser alguna de las siguientes constantes:

- CP_C_GROUP
- CP_C_LAYERS

INT DEFSHAPEF(INTEGER, INTEGER, FLOAT) Define la propiedad de un shape usando un flotante. Las propiedades de un shape que se pueden definir son:

- CP_C_E
- CP_C_U

FLOAT GETCPSHAPEF(INTEGER, INTEGER) Retorna los valores del shape que son flotantes. Campo puede ser alguna de las siguientes constantes:

- CP_C_E
- CP_C_U

INT DEFSHAPEP(INTEGER shape, INTEGER val, POINTER point) Define la propiedad de un shape usando un puntero. Las propiedades de un shape que se pueden definir son:

- CP_C_SURFACE_V

INT GETCPSHAPEP(INTEGER, INTEGER campo, POINTER point) Almacena en point alguno de los valores del shape. Point es el puntero a cpVect y campo puede ser:

- CP_C_SURFACE_V

INT CPPOLYSHAPEGETVERT(INTEGER shape, INTEGER n, POINTER point) Almacena en point el vértice número n del shape.

INT CPRECENTERPOLY(INTEGER n, POINTER v) Centra un polígono a (0,0). El parámetro n es el número de vértices y v es un arreglo de cpVects con la lista de vértices.

INT CPCENTROIDFORPOLY(INTEGER n, POINTER l, POINTER point) Calcula el centroide de un polígono. Los parámetros son el número de vértices, la lista de vértices (l es un arreglo de cpVects con la lista de vértices) y un puntero a cpVect para rellenar con el centroide.

6.5. Constraints

Todos los constraints tiene funciones para crearse y para modificarse una vez creados, debido a la gran cantidad de funciones reduciré la explicación para generalizar un poco.

Después de la descripción de la función que crea y agrega una constraint a un proceo, colocaré las funciones que permiten modificar u obtener sus propiedades.

En constraint iría el valor que la función add* arroja, en campo el valor que se quiere modificar o leer y en los siguientes parámetros el valor o los valores nuevos.

En el caso de los get*, si el 3ro y el 4to parámetro son punteros entonces hablamos de punteros a flotantes para rellenarlos con el valor del campo (no se acepta null en ninguno de los parámetros).

Los campos para las funciones set* o get* son:

- CP_C_ANCHR1
- CP_C_ANCHR2
- CP_C_DIST
- CP_C_MIN
- CP_C_MAX
- CP_C_GROOVEB
- CP_C_GROOVEA
- CP_C_RESTLENGTH
- CP_C_STIFFNESS
- CP_C_DAMPING
- CP_C_RESTANGLE
- CP_C_PHASE
- CP_C_RATCHET
- CP_C_RATIO
- CP_C_RATE

La mayoría de los Constraint tienen dos o tres campos de los anteriores sólamente.

6.5.1. FLOAT CPCONSTRAINTGETIMPULSE(INTEGER constraint)

retorna el último impulso que ha aplicado un constraint. Se debe recordar que phresolution indica el número de veces que el motor se ejecuta por frame, de tal forma que si phresolution es 3 (valor por defecto). Esta función retornará el 3er impulso.

6.5.2. INT DEFCONSTRAINTF(INTEGER constraint, INTEGER val, FLOAT v)

Define de un constraint el campo val con el flotante v. CP_C_CA , CP_C_CB refsec:cpcbb

6.5.3. FLOAT GETCONSTRAINTF(INTEGER, INTEGER)

Retorna alguno de los campos de la función anterior de un constraint.

6.5.4. INT DEFCONSTRAINTI(INTEGER, INTEGER, INTEGER)

Define de un constraint el campo val con el flotante v. CP_C_MAXFORCE , CP_C_BIASCOEF , CP_C_MAXBIAS ,

6.5.5. INT GETCONSTRAINTI(INTEGER, INTEGER, INTEGER)

Retorna alguno de los campos de la función anterior de un constraint.

Funciones de creación de constraints

6.5.6. INT ADDDAMPEDSPRING(INTEGER id1, INTEGER id2, FLOAT x1, FLOAT y1, FLOAT x2, FLOAT y2, FLOAT d, FLOAT r, FLOAT am)

Agrega una fuerza tipo resorte entre los procesos cuyas ids son id1 e id2 en los puntos (x1,y1) y (x2,y2). Las propiedades del resorte son:
d - distancia que idealmente debe haber entre los cuerpos
r - rigidez del resorte
am - amortiguación del resorte.

INT SETDAMPEDSPRINGPROPERTIES(INTEGER constraint, INTEGER campo, FLOAT x, FLOAT y)

INT GETDAMPEDSPRINGPROPERTIES(INTEGER constraint, INTEGER campo, POINTER x, POINTER y)

- CP_C_ANCHR1
- CP_C_ANCHR2

INT SETDAMPEDSPRINGPROPERTIES(INTEGER constraint, INTEGER campo, FLOAT a)

FLOAT GETDAMPEDSPRINGPROPERTIES(INTEGER constraint, INTEGER campo)

- CP_C_RESTLENGTH
- CP_C_STIFFNESS
- CP_C_DAMPING

6.5.7. INT ADDPIVOTJOINT(INTEGER id1, INTEGER id2, FLOAT x, FLOAT y)

Agrega una unión entre los procesos id1 e id2. La unión se hará en el punto dado por (x,y) en coordenadas globales.

6.5.8. INT ADDPIVOTJOINT2(INTEGER id1, INTEGER id2, FLOAT x1, FLOAT y1, FLOAT x2, FLOAT y2)

Agrega una unión entre los procesos id1 e id2. La unión se hará en los puntos dados por (x1,y1) y (x2,y2) en los cuerpos de los procesos (En coordenadas locales).

INT SETPIVOTJOINTPROPERTIES(INTEGER constraint, INTEGER campo, FLOAT x, FLOAT y)

INT GETPIVOTJOINTPROPERTIES(INTEGER constraint, INTEGER campo, POINTER x, POINTER y)

- CP_C_ANCHR
- CP_C_ANCHR2

6.5.9. INT ADDDAMPEDROTARYSPRING(INTEGER id1, INTEGER id2, FLOAT d, FLOAT r, FLOAT am)

Agrega un resorte de tipo angular entre los procesos id1 e id2. Las propiedades del resorte son:
d - distancia que idealmente debe haber entre los cuerpos
r - rigidez del resorte
am - amortiguación del resorte.

INT SETDAMPEDROTARYSPRINGPROPERTIES(INTEGER constraint, INTEGER campo, FLOAT a)

FLOAT GETDAMPEDROTARYSPRINGPROPERTIES(INTEGER constraint, INTEGER campo)

- CP_C_RESTLENGTH
- CP_C_STIFFNESS
- CP_C_DAMPING

6.5.10. INT ADDROTARYLIMITJOINT(INTEGER id1, INTEGER id2, FLOAT min, FLOAT max)

Limita la rotación relativa de los procesos id1 e id2 mediante min y max.

INT SETROTARYLIMITJOINTPROPERTIES(INTEGER constraint, INTEGER campo, FLOAT a)

FLOAT GETROTARYLIMITJOINTPROPERTIES(INTEGER constraint, INTEGER campo)

- CP_C_MIN
- CP_C_MAX

6.5.11. INT ADDRATCHETJOINT(INTEGER id1, INTEGER id2, FLOAT phase, FLOAT ratchet)

El api original describe el funcionamiento como: Works like a socket wrench. ratchet is the distance between textgravedbl clicks textacutedbl, phase is the initial offset to use when deciding where the ratchet angles are.

INT SETRATCHETJOINTPROPERTIES(INTEGER constraint, INTEGER campo, FLOAT a)

FLOAT GETRATCHETJOINTPROPERTIES(INTEGER constraint, INTEGER campo)

- CP_C_PHASE
- CP_C_RATCHET
- CP_C_ANGLE

6.5.12. INT ADDSIMPLEMOTOR(INTEGER id1, INTEGER id2, FLOAT rate)

Usando los procesos id1 e id2 genera el efecto de un motor. El api original dice: Keeps the relative angular velocity of a pair of bodies constant. rate is the desired relative angular velocity. You will usually want to set an force (torque) maximum for motors as otherwise they will be able to apply a nearly infinite torque to keep the bodies moving.

INT SETSIMPLEMOTORPROPERTIES(INTEGER constraint, INTEGER campo, FLOAT a)

FLOAT GETSIMPLEMOTORPROPERTIES(INTEGER constraint, INTEGER campo)

- CP_C_RATE

6.5.13. INT ADDGROOVEJOINT(INTEGER id1, INTEGER id2, FLOAT groove_a_x, groove_a_y, FLOAT groove_b_x, groove_b_y, FLOAT anchr2_x, anchr2_y)

El api original dice:

The groove goes from groov_a to groove_b on body a, and the pivot is attached to anchr2 on body b. All coordinates are body local.

INT SETGROOVEJOINTPROPERTIES(INTEGER constraint, INTEGER campo, FLOAT x, FLOAT y)

INT GETGROOVEJOINTPROPERTIES(INTEGER constraint, INTEGER campo, POINTER x, POINTER y)

- P_C_ANCHR2
- CP_C_GROOVEB
- CP_C_GROOVEA

6.5.14. INT ADDSLIDEJOINT(INTEGER id1, INTEGER id2, FLOAT x1, FLOAT y1, FLOAT x2, FLOAT y2, FLOAT min, FLOAT max)

Min y max es la distancia permitida entre los puntos dados por (x1, y1) y el proceso cuya id es id1, y el punto (x2,y2) y el proceso de id2.

INT SETSLIDEJOINTPROPERTIES(INTEGER constraint, INTEGER campo, FLOAT x, FLOAT y)

INT GETSLIDEJOINTPROPERTIES(INTEGER constraint, INTEGER campo, POINTER x, POINTER y)

- CP_C_ANCHR1
- CP_C_ANCHR2

INT SETSLIDEJOINTPROPERTIES(INTEGER constraint, INTEGER campo, FLOAT a)

FLOAT GETSLIDEJOINTPROPERTIES(INTEGER constraint, INTEGER)

- CP_C_MIN
- CP_C_MAX

6.5.15. INT ADDPINJOINT(INTEGER id1, INTEGER id2, FLOAT x1, FLOAT y1, FLOAT x2, FLOAT y2)

Agrega una unión entre los procesos id1 e id2 En los puntos (x1, y1) y (x2,y2) con la distancia que hay entre los procesos a la hora de crearse el constraint.

INT SETPINJOINTPROPERTIES(INTEGER constraint , INTEGER campor, FLOAT x, FLOAT y)

INT GETPINJOINTPROPERTIES(INTEGER constraint, INTEGER campo, POINTER, POINTER)

- CP_C_ANCHR1
- CP_C_ANCHR2

6.5.16. INT SETPINJOINTPROPERTIES(INTEGER constraint, INTEGER campo, FLOAT a)

6.5.17. FLOAT GETPINJOINTPROPERTIES(INTEGER constraint, INTEGER)

- CP_C_DIST

6.5.18. INT ADDGEARJOINT(INTEGER constraint, INTEGER campo, FLOAT Phase, FLOAT Ratio)

Agrega un joint tipo engrane entre dos procesos. La fase es la inicial diferencia de ángulos entre los dos procesos y el ratio es la relación entre las velocidades angulares.

INT SETGEARJOINTPROPERTIES(INTEGER constraint, INTEGER campo, FLOAT a)

FLOAT GETGEARJOINTPROPERTIES(INTEGER constraint, INTEGER campo)

- CP_C_RATIO
- CP_C_PHASE

6.5.19. INT REMOVECONSTRAINT(INTEGER id, INTEGER constraint)

Elimina un constraint. Los parámetros son el id de alguno de los procesos en los que actua éste y el constraint.

6.5.20. INT GETCONSTRAINTS(INTEGER)

Retorna un contraint que se ha agragado al proceso cada vez que sea llamada la función hasta o cero si no hay más constraints.

La función se reinicia automáticamente cada frame.

Con esta función se puede recorrer la lista de constraints para buscar alguno en específico o para hacer algo con todos ellos.

Funciones inseguras

Estas funciones pueden desestabilizar la simulación, pero nunca generarán problemas de otro tipo.

6.5.21. INT SETENDPOINTSLINE(FLOAT x1, FLOAT y1, FLOAT x2, FLOAT y2)

Estas funciones permiten modificar el shape de tipo línea del proceso. Los parámetros son el principio de la linea (x,y) y el final de la línea (x,y).

6.5.22. INT SETRADIUSLINE(FLOAT r)

Modifica el radio de un shape tipo línea.

6.5.23. INT SETOFFSETCIRCLE(FLOAT x, FLOAT y)

Modifica la posición con respecto del centro del proceso del shape círculo. Sus parámetros son el (x,y) del nuevo punto central.

6.5.24. INT SETRADIUSCIRCLE(FLOAT radio)

Modifica el radio de un shape tipo círculo.

6.5.25. INT SETVERTCONVEXPOLIGON(FLOAT x, FLOAT y, INTEGER n, POINTER l)

Modifica la forma de un polígono convexo, los parámetros son las coordenasas x e y del centro del polígono (coordenadas locales al body), el número de vértices y una lista de cpVect con los vértices.

6.6. Heredadas primitivas

Estas funciones usan en body la local body del proceso y en shape la local shape de él. La mayoría de ellas tienen la contraparte que funciona con el id del proceso.

6.6.1. INT CPBODYUPDATEPOSITION(INTEGER body, FLOAT dt)

Actualiza la posición del body usando integración de Euler. Dt es interval/phresolution o cualquier otro valor que se necesite.

6.6.2. INT CPBODYRESETFORCES(INTEGER body)

Reinicia las fuerzas del body.

6.6.3. INT CPBODYSLEEPWITHGROUP(INTEGER, INTEGER)

Actualmente no funciona.

6.6.4. INT CPBODYAPPLYFORCE(INTEGER body, POINTER f, POINTER point)

Aplica la fuerza f (cpvect) al body en el punto local point (cpvect).

6.6.5. INT CPBODYAPPLYIMPULSE(INTEGER body, POINTER i, POINTER point)

Aplica el impulso i (cpvect) al body en el punto local point (cpvect). El impulso no tiene en cuenta la actual rotación.

6.6.6. INT CPBODYSLEW(INTEGER, POINTER, FLOAT)

Mueve el proceso dueño del body al punto p (cpVect) en el tiempo dt (dt puede ser igual a interval/phresolution). Usar esta función requiere también usar la función CPBODYUPDATEVELOCITY.

INT CPBODYUPDATEVELOCITY(INTEGER body, POINTER gravedad, FLOAT damping, FLOAT dt) Actualiza la velocidad de un proceso. Los parámetros son el body, un cpVect con la gravedad, el damping del mundo y el dt (dt puede ser igual a interval/phresolution).

INT SETENDPOINTSLINE(INTEGER shape,FLOAT x1, FLOAT y1, FLOAT x2, FLOAT y2) Estas funciones permiten modificar el shape de tipo línea del proceso. Los parámetros son el principio de la linea (x,y) y el final de la línea (x,y).

6.6.7. INT SETRADIUSLINEI(INTEGER shape,FLOAT,r)

Modifica el radio de un shape tipo línea.

6.6.8. INT SETOFFSETCIRCLEI(INTEGER shape,FLOAT x, FLOAT y)

Modifica la posición con respecto del centro del proceso del shape círculo. Sus parámetros son el (x,y) del nuevo punto central.

6.6.9. INT SETRADIUSCIRCLEI(INTEGER shape,FLOAT radio)

Modifica el radio de un shape tipo círculo.

6.6.10. INT SETVERTCONVEXPOLIGONI(INTEGER shape,FLOAT x, FLOAT y, INTEGER n, POINTER l)

Modifica la forma de un polígono convexo, los parámetros son las coordenasas x e y del centro del polígono (coordenadas locales al body), el número de vértices y una lista de cpVect con los vértices.

6.6.11. FLOAT CPSEGMENTSHAPEGETRADIUS(INTEGER shape)

Retorna el radio de un shape de tipo línea shape.

6.6.12. FLOAT CPCIRCLESHAPEGETRADIUS(INTEGER shape)

Retorna el radio de un shape de tipo círculo.

6.6.13. INT CPSEGMENTSHAPEGETNORMAL(INTEGER shape, POINTER normal)

Almacena en el cpVect normal la normal de un shape de tipo línea.

6.6.14. INT CPCIRCLESHAPEGETOFFSET(INTEGER shape, POINTER point)

Almacena en el cpVect point el desplazamiento del shape (en coordenadas locales).

INT CPSEGMENTSHAPEGETA(INTEGER shape, POINTER point) Almacena en el cpVect point el punto A de un shape de tipo línea.

6.6.15. INT CPSEGMENTSHAPEGETB(INTEGER shape, POINTER point)

Almacena en el cpVect point el punto B de un shape de tipo línea.

INT CPPOLYSHAPEGETNUMVERTS(INTEGER shape) Retorna el número de vértices del shape.

INT CPSHAPECACHEBB(INTEGER shape, POINTER boundingBox) Almacena en boundingBox el boundingBox del shape.

6.7. Miscelanea

6.7.1. INT CLEANSPACE()

Elimina todos los cuerpos físicos y todo el mundo físico. Esta función debe ser llamada después de eliminar todos los procesos. Por lo general se encarga de eliminar los shapes agragados al fondo.

Al finalizar un proceso o cerrar el programa se eliminan los cuerpos y se libera la memoria automáticamente.

6.7.2. STRING CPVSTR(POINTER a)

Retorna una cadena que representa al vector a. Sirve para usarse con say o write.

6.7.3. POINTER CPV(FLOAT a, FLOAT b, POINTER res)

Rellena res con (a,b).

6.7.4. INT CPBODYLOCAL2WORLD(INTEGER body, POINTER a, POINTER b)

Transforma el vector a de cordenadas locales a uno de cordenadas en el scroll o background (coordenadas globales) usando las coordenadas del body y rellena del cpvect b con el resultado.

6.7.5. INT CPBODYWORLD2LOCAL(INTEGER body, POINTER v, POINTER a)

Transforma un vector v de coordenadas globale a locales del body.

6.7.6. INT LOCAL2WORLD(INTEGER id, FLOAT x1, FLOAT y1, POINTER x2, POINTER y2)

Almacena en x2 y y2 las coordenadas globales obtenidas de x1 e y1 usando el id de algún proceso.

6.7.7. INT WORLD2LOCAL(INTEGER id, FLOAT x1, FLOAT y1, POINTER x2, POINTER y2)

6.7.8. INT CPRESETSHAPEIDCOUNTER()

Reinicia el índice del creador de shapes.