

My Project

Generated by Doxygen 1.8.9.1

Mon Jul 27 2015 14:59:41

Contents

1	Module Index	1
1.1	Modules	1
2	Class Index	3
2.1	Class List	3
3	Module Documentation	5
3.1	Misc	5
3.1.1	Detailed Description	6
3.1.2	Macro Definition Documentation	6
3.1.2.1	CP_CONVEX_HULL	6
3.1.3	Function Documentation	7
3.1.3.1	cpAreaForCircle	7
3.1.3.2	cpAreaForPoly	7
3.1.3.3	cpConvexHull	7
3.1.3.4	cpMomentForCircle	7
3.1.3.5	cpMomentForSegment	7
3.2	Basic Types	8
3.2.1	Detailed Description	9
3.2.2	Typedef Documentation	9
3.2.2.1	cpCollisionID	9
3.2.2.2	cpFloat	9
3.3	Chipmunk Unsafe Shape Operations	10
3.3.1	Detailed Description	10
3.4	cpArbiter	11
3.4.1	Detailed Description	12
3.4.2	Function Documentation	12
3.4.2.1	cpArbiterCallWildcardBeginA	12
3.4.2.2	cpArbiterCallWildcardBeginB	13
3.4.2.3	cpArbiterCallWildcardPreSolveA	13
3.4.2.4	cpArbiterCallWildcardPreSolveB	13
3.4.2.5	cpArbiterGetBodies	13

3.4.2.6	cpArbiterGetShapes	13
3.4.2.7	cpArbiterIgnore	13
3.4.2.8	cpArbiterSetContactPointSet	13
3.4.2.9	cpArbiterSetUserData	13
3.4.2.10	cpArbiterTotalImpulse	14
3.4.2.11	cpArbiterTotalKE	14
3.5	cpBB	15
3.5.1	Detailed Description	15
3.6	cpBody	16
3.6.1	Detailed Description	18
3.6.2	Enumeration Type Documentation	18
3.6.2.1	cpBodyType	18
3.6.3	Function Documentation	19
3.6.3.1	cpBodySetPositionUpdateFunc	19
3.7	cpConstraint	20
3.7.1	Detailed Description	21
3.7.2	Function Documentation	21
3.7.2.1	cpConstraintSetErrorBias	21
3.8	cpDampedRotarySpring	22
3.8.1	Detailed Description	22
3.9	cpDampedSpring	23
3.9.1	Detailed Description	23
3.10	cpGearJoint	24
3.10.1	Detailed Description	24
3.11	cpGrooveJoint	25
3.11.1	Detailed Description	25
3.12	cpPinJoint	26
3.12.1	Detailed Description	26
3.13	cpPivotJoint	27
3.13.1	Detailed Description	27
3.14	cpPolyShape	28
3.14.1	Detailed Description	28
3.14.2	Function Documentation	28
3.14.2.1	cpPolyShapeInit	28
3.14.2.2	cpPolyShapeInitRaw	29
3.14.2.3	cpPolyShapeNew	29
3.14.2.4	cpPolyShapeNewRaw	29
3.15	cpRatchetJoint	30
3.15.1	Detailed Description	30
3.16	cpRotaryLimitJoint	31

3.16.1 Detailed Description	31
3.17 cpShape	32
3.17.1 Detailed Description	34
3.17.2 Function Documentation	34
3.17.2.1 cpShapePointQuery	34
3.17.2.2 cpShapeSetBody	34
3.18 cpCircleShape	35
3.19 cpSegmentShape	36
3.20 cpSimpleMotor	37
3.20.1 Detailed Description	37
3.21 cpSlideJoint	38
3.21.1 Detailed Description	38
3.22 cpSpace	39
3.22.1 Detailed Description	42
3.22.2 Typedef Documentation	42
3.22.2.1 cpCollisionBeginFunc	42
3.22.2.2 cpCollisionPreSolveFunc	43
3.22.3 Function Documentation	43
3.22.3.1 cpSpaceAddCollisionHandler	43
3.22.3.2 cpSpaceAddPostStepCallback	43
3.22.3.3 cpSpaceAddShape	43
3.22.3.4 cpSpaceBBQuery	43
3.22.3.5 cpSpaceGetCollisionBias	43
3.22.3.6 cpSpaceGetCollisionPersistence	43
3.22.3.7 cpSpaceGetCollisionSlop	43
3.22.3.8 cpSpaceGetCurrentTimeStep	44
3.22.3.9 cpSpaceGetDamping	44
3.22.3.10 cpSpaceGetIdleSpeedThreshold	44
3.22.3.11 cpSpaceGetSleepTimeThreshold	44
3.22.3.12 cpSpaceGetStaticBody	44
3.22.3.13 cpSpaceGetUserData	44
3.22.3.14 cpSpaceSetCollisionBias	44
3.22.3.15 cpSpaceSetCollisionPersistence	44
3.22.3.16 cpSpaceSetCollisionSlop	45
3.22.3.17 cpSpaceSetDamping	45
3.22.3.18 cpSpaceSetIdleSpeedThreshold	45
3.22.3.19 cpSpaceSetSleepTimeThreshold	45
3.22.3.20 cpSpaceSetUserData	45
3.23 cpSpatialIndex	46
3.23.1 Detailed Description	48

3.23.2	Typedef Documentation	48
3.23.2.1	cpBBTreeVelocityFunc	48
3.23.2.2	cpSpatialIndexBBFunc	48
3.23.3	Function Documentation	48
3.23.3.1	cpSpaceHashResize	48
3.23.3.2	cpSpatialIndexContains	48
3.23.3.3	cpSpatialIndexInsert	48
3.23.3.4	cpSpatialIndexReindexQuery	48
3.23.3.5	cpSpatialIndexRemove	49
3.24	cpVect	50
3.24.1	Detailed Description	51
3.24.2	Function Documentation	51
3.24.2.1	cpvcross	51
3.25	cpMat2x2	52
3.25.1	Detailed Description	52
4	Class Documentation	53
4.1	cpArbiter Struct Reference	53
4.1.1	Detailed Description	53
4.2	cpArbiterThread Struct Reference	53
4.2.1	Detailed Description	54
4.3	cpArray Struct Reference	54
4.3.1	Detailed Description	54
4.4	cpBB Struct Reference	54
4.4.1	Detailed Description	54
4.5	cpBody Struct Reference	55
4.5.1	Detailed Description	55
4.6	cpCircleShape Struct Reference	55
4.6.1	Detailed Description	56
4.7	cpCollisionHandler Struct Reference	56
4.7.1	Detailed Description	56
4.7.2	Member Data Documentation	56
4.7.2.1	postSolveFunc	56
4.7.2.2	preSolveFunc	56
4.7.2.3	typeA	57
4.7.2.4	typeB	57
4.8	cpCollisionInfo Struct Reference	57
4.8.1	Detailed Description	57
4.9	cpConstraint Struct Reference	57
4.9.1	Detailed Description	58

4.10	cpConstraintClass Struct Reference	58
4.10.1	Detailed Description	58
4.11	cpContact Struct Reference	58
4.11.1	Detailed Description	59
4.12	cpContactPointSet Struct Reference	59
4.12.1	Detailed Description	59
4.12.2	Member Data Documentation	59
4.12.2.1	distance	59
4.13	cpDampedRotarySpring Struct Reference	60
4.13.1	Detailed Description	60
4.14	cpDampedSpring Struct Reference	60
4.14.1	Detailed Description	61
4.15	cpGearJoint Struct Reference	61
4.15.1	Detailed Description	61
4.16	cpGrooveJoint Struct Reference	61
4.16.1	Detailed Description	62
4.17	cpMat2x2 Struct Reference	62
4.17.1	Detailed Description	62
4.18	cpPinJoint Struct Reference	62
4.18.1	Detailed Description	62
4.19	cpPivotJoint Struct Reference	63
4.19.1	Detailed Description	63
4.20	cpPointQueryInfo Struct Reference	63
4.20.1	Detailed Description	63
4.20.2	Member Data Documentation	64
4.20.2.1	gradient	64
4.21	cpPolyline Struct Reference	64
4.22	cpPolylineSet Struct Reference	64
4.22.1	Detailed Description	64
4.23	cpPolyShape Struct Reference	64
4.23.1	Detailed Description	65
4.24	cpPostStepCallback Struct Reference	65
4.25	cpRatchetJoint Struct Reference	65
4.25.1	Detailed Description	65
4.26	cpRotaryLimitJoint Struct Reference	66
4.26.1	Detailed Description	66
4.27	cpSegmentQueryInfo Struct Reference	66
4.27.1	Detailed Description	66
4.28	cpSegmentShape Struct Reference	66
4.28.1	Detailed Description	67

4.29	cpShape Struct Reference	67
4.29.1	Detailed Description	67
4.30	cpShapeClass Struct Reference	68
4.30.1	Detailed Description	68
4.31	cpShapeFilter Struct Reference	68
4.31.1	Detailed Description	68
4.31.2	Member Data Documentation	68
4.31.2.1	categories	68
4.31.2.2	group	68
4.31.2.3	mask	69
4.32	cpShapeMassInfo Struct Reference	69
4.32.1	Detailed Description	69
4.33	cpSimpleMotor Struct Reference	69
4.33.1	Detailed Description	69
4.34	cpSlideJoint Struct Reference	70
4.34.1	Detailed Description	70
4.35	cpSpace Struct Reference	70
4.35.1	Detailed Description	71
4.36	cpSpaceDebugColor Struct Reference	71
4.36.1	Detailed Description	71
4.37	cpSpaceDebugDrawOptions Struct Reference	71
4.37.1	Detailed Description	72
4.38	cpSpatialIndexClass Struct Reference	72
4.38.1	Detailed Description	73
4.39	cpSplittingPlane Struct Reference	73
4.39.1	Detailed Description	73
4.40	cpTransform Struct Reference	73
4.40.1	Detailed Description	73
4.41	cpVect Struct Reference	74
4.41.1	Detailed Description	74
Index		75

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Misc	5
Basic Types	8
Chipmunk Unsafe Shape Operations	10
cpArbiter	11
cpBB	15
cpBody	16
cpConstraint	20
cpDampedRotarySpring	22
cpDampedSpring	23
cpGearJoint	24
cpGrooveJoint	25
cpPinJoint	26
cpPivotJoint	27
cpPolyShape	28
cpRatchetJoint	30
cpRotaryLimitJoint	31
cpShape	32
cpCircleShape	35
cpSegmentShape	36
cpSimpleMotor	37
cpSlideJoint	38
cpSpace	39
cpSpatialIndex	46
cpVect	50
cpMat2x2	52

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

cpArbiter	Tracks pairs of colliding shapes	53
cpArbiterThread	A doubly linked list for the cpArbiter values	53
cpArray	Chipmunk's array data structure	54
cpBB	Chipmunk's axis-aligned 2D bounding box type. (left, bottom, right, top)	54
cpBody	Chipmunk's rigid body type	55
cpCircleShape	A perfect circle shape	55
cpCollisionHandler	Struct that holds function callback pointers to configure custom collision handling	56
cpCollisionInfo	Holds information about the collision	57
cpConstraint	Constraints connect two cpBody objects together	57
cpConstraintClass	Struct that holds function callback pointers for constraints	58
cpContact	Holds information about the contact points of the collision	58
cpContactPointSet	A struct that wraps up the important collision data for an arbiter	59
cpDampedRotarySpring	Like a cpDampedSpring , but operates in a rotational fashion	60
cpDampedSpring	A spring with a damper	60
cpGearJoint	Maintains a specific angular velocity between the two bodies	61
cpGrooveJoint	Similar to a pivot joint, but one of the anchors is a line segment that the pivot can slide in	61
cpMat2x2	2x2 matrix type used for tensors and such	62
cpPinJoint	The two anchor points are always the same distance apart	62
cpPivotJoint	Pivot joints hold two points on two bodies together allowing them to rotate freely around the pivot	63

cpPointQueryInfo	
Point query info struct	63
cpPolyline	64
cpPolylineSet	
Polyline sets are collections of polylines, generally built by cpMarchSoft() or cpMarchHard() . . .	64
cpPolyShape	
A convex polygon shape	64
cpPostStepCallback	65
cpRatchetJoint	
Create rotary ratches similar to a socket wrench	65
cpRotaryLimitJoint	
Constrains the bodies' orientations to be within a certain angle of each other	66
cpSegmentQueryInfo	
Segment query info struct	66
cpSegmentShape	
A beveled (rounded) segment shape	66
cpShape	
The cpShape struct defines the shape of a rigid body	67
cpShapeClass	
Struct that holds function callback pointers for shapes	68
cpShapeFilter	
Fast collision filtering type that is used to determine if two objects collide before calling collision or query callbacks	68
cpShapeMassInfo	
Struct that holds information about the mass of the shape	69
cpSimpleMotor	
Maintains a specific angular relative velocity between two objects	69
cpSlideJoint	
Slide joints hold the distance between points on two bodies between a minimum and a maximum	70
cpSpace	
Containers for simulating objects in Chipmunk	70
cpSpaceDebugColor	
Color type to use with the space debug drawing API	71
cpSpaceDebugDrawOptions	
Struct used with cpSpaceDebugDraw() containing drawing callbacks and other drawing settings	71
cpSpatialIndexClass	
Used to accelerate collision detection	72
cpSplittingPlane	
Splitting plane	73
cpTransform	
Column major affine transform	73
cpVect	
Chipmunk's 2D vector type	74

Chapter 3

Module Documentation

3.1 Misc

A set of miscellaneous functions for calculating the area, moment of inertia and other properties of shapes.

Macros

- `#define CP_BUFFER_BYTES (32*1024)`
Allocated size for various Chipmunk buffers.
- `#define cpalloc calloc`
Chipmunk calloc() alias.
- `#define cprealloc realloc`
Chipmunk realloc() alias.
- `#define cpfree free`
Chipmunk free() alias.
- `#define CP_VERSION_MAJOR 7`
- `#define CP_VERSION_MINOR 0`
- `#define CP_VERSION_RELEASE 0`
- `#define CP_CONVEX_HULL(__count__, __verts__, __count_var__, __verts_var__)`
Convenience macro to work with cpConvexHull.

Typedefs

- `typedef struct cpArray cpArray`
- `typedef struct cpHashSet cpHashSet`
- `typedef struct cpBody cpBody`
- `typedef struct cpShape cpShape`
- `typedef struct cpCircleShape cpCircleShape`
- `typedef struct cpSegmentShape cpSegmentShape`
- `typedef struct cpPolyShape cpPolyShape`
- `typedef struct cpConstraint cpConstraint`
- `typedef struct cpPinJoint cpPinJoint`
- `typedef struct cpSlideJoint cpSlideJoint`
- `typedef struct cpPivotJoint cpPivotJoint`
- `typedef struct cpGrooveJoint cpGrooveJoint`
- `typedef struct cpDampedSpring cpDampedSpring`
- `typedef struct cpDampedRotarySpring cpDampedRotarySpring`
- `typedef struct cpRotaryLimitJoint cpRotaryLimitJoint`

- typedef struct [cpRatchetJoint](#) **cpRatchetJoint**
- typedef struct [cpGearJoint](#) **cpGearJoint**
- typedef struct [cpSimpleMotorJoint](#) **cpSimpleMotorJoint**
- typedef struct [cpCollisionHandler](#) **cpCollisionHandler**
- typedef struct [cpContactPointSet](#) **cpContactPointSet**
- typedef struct [cpArbiter](#) **cpArbiter**
- typedef struct [cpSpace](#) **cpSpace**

Functions

- [cpFloat cpMomentForCircle](#) ([cpFloat](#) m, [cpFloat](#) r1, [cpFloat](#) r2, [cpVect](#) offset)
Calculate the moment of inertia for a circle.
- [cpFloat cpAreaForCircle](#) ([cpFloat](#) r1, [cpFloat](#) r2)
Calculate area of a hollow circle.
- [cpFloat cpMomentForSegment](#) ([cpFloat](#) m, [cpVect](#) a, [cpVect](#) b, [cpFloat](#) radius)
Calculate the moment of inertia for a line segment.
- [cpFloat cpAreaForSegment](#) ([cpVect](#) a, [cpVect](#) b, [cpFloat](#) radius)
Calculate the area of a fattened (capsule shaped) line segment.
- [cpFloat cpMomentForPoly](#) ([cpFloat](#) m, int count, const [cpVect](#) *verts, [cpVect](#) offset, [cpFloat](#) radius)
Calculate the moment of inertia for a solid polygon shape assuming it's center of gravity is at it's centroid. The offset is added to each vertex.
- [cpFloat cpAreaForPoly](#) (const int count, const [cpVect](#) *verts, [cpFloat](#) radius)
Calculate the signed area of a polygon.
- [cpVect cpCentroidForPoly](#) (const int count, const [cpVect](#) *verts)
Calculate the natural centroid of a polygon.
- [cpFloat cpMomentForBox](#) ([cpFloat](#) m, [cpFloat](#) width, [cpFloat](#) height)
Calculate the moment of inertia for a solid box.
- [cpFloat cpMomentForBox2](#) ([cpFloat](#) m, [cpBB](#) box)
Calculate the moment of inertia for a solid box.
- int [cpConvexHull](#) (int count, const [cpVect](#) *verts, [cpVect](#) *result, int *first, [cpFloat](#) tol)
Calculate the convex hull of a given set of points.
- static [cpVect cpClosestPointOnSegment](#) (const [cpVect](#) p, const [cpVect](#) a, const [cpVect](#) b)
Returns the closest point on the line segment ab, to the point p.

Variables

- const char * [cpVersionString](#)
Version string.

3.1.1 Detailed Description

A set of miscellaneous functions for calculating the area, moment of inertia and other properties of shapes.

3.1.2 Macro Definition Documentation

3.1.2.1 #define CP_CONVEX_HULL(__count__, __verts__, __count_var__, __verts_var__)

Value:

```
cpVect *__verts_var__ = (cpVect *)alloca(__count__*sizeof(cpVect)); \
int __count_var__ = cpConvexHull(__count__, __verts__, __verts_var__, NULL, 0.0); \
```

Convenience macro to work with `cpConvexHull`.

`count` and `verts` is the input array passed to `cpConvexHull()`. `count_var` and `verts_var` are the names of the variables the macro creates to store the result. The output vertex array is allocated on the stack using `alloca()` so it will be freed automatically, but cannot be returned from the current scope.

3.1.3 Function Documentation

3.1.3.1 `cpFloat cpAreaForCircle (cpFloat r1, cpFloat r2)`

Calculate area of a hollow circle.

`r1` and `r2` are the inner and outer diameters. A solid circle has an inner diameter of 0.

3.1.3.2 `cpFloat cpAreaForPoly (const int count, const cpVect * verts, cpFloat radius)`

Calculate the signed area of a polygon.

A Clockwise winding gives positive area. This is probably backwards from what you expect, but matches Chipmunk's the winding for poly shapes.

3.1.3.3 `int cpConvexHull (int count, const cpVect * verts, cpVect * result, int * first, cpFloat tol)`

Calculate the convex hull of a given set of points.

Returns the count of points in the hull. `result` must be a pointer to a `cpVect` array with at least `count` elements. If `verts == result`, then `verts` will be reduced inplace. `first` is an optional pointer to an integer to store where the first vertex in the hull came from (i.e. `verts[first] == result[0]`) `tol` is the allowed amount to shrink the hull when simplifying it. A tolerance of 0.0 creates an exact hull.

3.1.3.4 `cpFloat cpMomentForCircle (cpFloat m, cpFloat r1, cpFloat r2, cpVect offset)`

Calculate the moment of inertia for a circle.

`r1` and `r2` are the inner and outer diameters. A solid circle has an inner diameter of 0.

3.1.3.5 `cpFloat cpMomentForSegment (cpFloat m, cpVect a, cpVect b, cpFloat radius)`

Calculate the moment of inertia for a line segment.

Beveling radius is not supported.

3.2 Basic Types

Most of these types can be configured at compile time.

Macros

- `#define cpfsqrt sqrt`
- `#define cpfsin sin`
- `#define cpfcos cos`
- `#define cpfacos acos`
- `#define cpfatan2 atan2`
- `#define cpfmod fmod`
- `#define cpfexp exp`
- `#define cpfpow pow`
- `#define cpffloor floor`
- `#define cpfceil ceil`
- `#define CPFLOAT_MIN DBL_MIN`
- `#define INFINITY (1e1000)`
- `#define M_PI 3.14159265358979323846264338327950288`
- `#define M_E 2.71828182845904523536028747135266250`
- `#define cpTrue 1`
true value.
- `#define cpFalse 0`
false value.
- `#define CP_NO_GROUP ((cpGroup)0)`
Value for cpShape.group signifying that a shape is in no group.
- `#define CP_ALL_CATEGORIES (~(cpBitmask)0)`
Value for cpShape.layers signifying that a shape is in every layer.
- `#define CP_WILDCARD_COLLISION_TYPE (~(cpCollisionType)0)`
cpCollisionType value internally reserved for hashing wildcard handlers.

Typedefs

- `typedef double cpFloat`
Chipmunk's floating point type.
- `typedef uintptr_t cpHashValue`
Hash value type.
- `typedef uint32_t cpCollisionID`
Type used internally to cache colliding object info for cpCollideShapes().
- `typedef unsigned char cpBool`
Chipmunk's boolean type.
- `typedef void * cpDataPointer`
Type used for user data pointers.
- `typedef uintptr_t cpCollisionType`
Type used for cpSpace.collision_type.
- `typedef uintptr_t cpGroup`
Type used for cpShape.group.
- `typedef unsigned int cpBitmask`
Type used for cpShapeFilter category and mask.
- `typedef unsigned int cpTimestamp`
Type used for various timestamps in Chipmunk.

Functions

- static `cpFloat cpfmax (cpFloat a, cpFloat b)`
Return the max of two cpFloats.
- static `cpFloat cpfmin (cpFloat a, cpFloat b)`
Return the min of two cpFloats.
- static `cpFloat cpfabs (cpFloat f)`
Return the absolute value of a cpFloat.
- static `cpFloat cpfclamp (cpFloat f, cpFloat min, cpFloat max)`
Clamp f to be between min and max .
- static `cpFloat cpfclamp01 (cpFloat f)`
Clamp f to be between 0 and 1.
- static `cpFloat cpflerp (cpFloat f1, cpFloat f2, cpFloat t)`
Linearly interpolate (or extrapolate) between $f1$ and $f2$ by t percent.
- static `cpFloat cpflerpconst (cpFloat f1, cpFloat f2, cpFloat d)`
Linearly interpolate from $f1$ to $f2$ by no more than d .

3.2.1 Detailed Description

Most of these types can be configured at compile time.

3.2.2 Typedef Documentation

3.2.2.1 `typedef uint32_t cpCollisionID`

Type used internally to cache colliding object info for `cpCollideShapes()`.

Should be at least 32 bits.

3.2.2.2 `typedef double cpFloat`

Chipmunk's floating point type.

Can be reconfigured at compile time.

3.3 Chipmunk Unsafe Shape Operations

These functions are used for mutating collision shapes.

Functions

- void [cpCircleShapeSetRadius](#) ([cpShape](#) *shape, [cpFloat](#) radius)
Set the radius of a circle shape.
- void [cpCircleShapeSetOffset](#) ([cpShape](#) *shape, [cpVect](#) offset)
Set the offset of a circle shape.
- void [cpSegmentShapeSetEndpoints](#) ([cpShape](#) *shape, [cpVect](#) a, [cpVect](#) b)
Set the endpoints of a segment shape.
- void [cpSegmentShapeSetRadius](#) ([cpShape](#) *shape, [cpFloat](#) radius)
Set the radius of a segment shape.
- void [cpPolyShapeSetVerts](#) ([cpShape](#) *shape, int count, [cpVect](#) *verts, [cpTransform](#) transform)
Set the vertexes of a poly shape.
- void [cpPolyShapeSetVertsRaw](#) ([cpShape](#) *shape, int count, [cpVect](#) *verts)
- void [cpPolyShapeSetRadius](#) ([cpShape](#) *shape, [cpFloat](#) radius)
Set the radius of a poly shape.

3.3.1 Detailed Description

These functions are used for mutating collision shapes.

Chipmunk does not have any way to get velocity information on changing shapes, so the results will be unrealistic. You must explicitly include the [chipmunk_unsafe.h](#) header to use them.

3.4 cpArbiter

The `cpArbiter` struct tracks pairs of colliding shapes.

Classes

- struct `cpContactPointSet`
A struct that wraps up the important collision data for an arbiter.

Macros

- `#define CP_MAX_CONTACTS_PER_ARBITER 2`
- `#define CP_ARBITER_GET_SHAPES(__arb__, __a__, __b__) cpShape *__a__, *__b__; cpArbiterGetShapes(__arb__, &__a__, &__b__);`
A macro shortcut for defining and retrieving the shapes from an arbiter.
- `#define CP_ARBITER_GET_BODIES(__arb__, __a__, __b__) cpBody *__a__, *__b__; cpArbiterGetBodies(__arb__, &__a__, &__b__);`
A macro shortcut for defining and retrieving the bodies from an arbiter.

Functions

- `cpFloat cpArbiterGetRestitution (const cpArbiter *arb)`
Get the restitution (elasticity) that will be applied to the pair of colliding objects.
- `void cpArbiterSetRestitution (cpArbiter *arb, cpFloat restitution)`
Override the restitution (elasticity) that will be applied to the pair of colliding objects.
- `cpFloat cpArbiterGetFriction (const cpArbiter *arb)`
Get the friction coefficient that will be applied to the pair of colliding objects.
- `void cpArbiterSetFriction (cpArbiter *arb, cpFloat friction)`
Override the friction coefficient that will be applied to the pair of colliding objects.
- `cpVect cpArbiterGetSurfaceVelocity (cpArbiter *arb)`
- `void cpArbiterSetSurfaceVelocity (cpArbiter *arb, cpVect vr)`
- `cpDataPointer cpArbiterGetUserData (const cpArbiter *arb)`
Get the user data pointer associated with this pair of colliding objects.
- `void cpArbiterSetUserData (cpArbiter *arb, cpDataPointer userData)`
Set a user data point associated with this pair of colliding objects.
- `cpVect cpArbiterTotalImpulse (const cpArbiter *arb)`
Calculate the total impulse including the friction that was applied by this arbiter.
- `cpFloat cpArbiterTotalKE (const cpArbiter *arb)`
Calculate the amount of energy lost in a collision including static, but not dynamic friction.
- `cpBool cpArbiterIgnore (cpArbiter *arb)`
Mark a collision pair to be ignored until the two objects separate.
- `void cpArbiterGetShapes (const cpArbiter *arb, cpShape **a, cpShape **b)`
Return the colliding shapes involved for this arbiter.
- `void cpArbiterGetBodies (const cpArbiter *arb, cpBody **a, cpBody **b)`
Return the colliding bodies involved for this arbiter.
- `cpContactPointSet cpArbiterGetContactPointSet (const cpArbiter *arb)`
Return a contact set from an arbiter.
- `void cpArbiterSetContactPointSet (cpArbiter *arb, cpContactPointSet *set)`
Replace the contact point set for an arbiter.
- `cpBool cpArbiterIsFirstContact (const cpArbiter *arb)`

- Returns true if this is the first step a pair of objects started colliding.*
- `cpBool cpArbiterIsRemoval` (const `cpArbiter` *arb)
 - Returns true if the separate callback is due to a shape being removed from the space.*
- `int cpArbiterGetCount` (const `cpArbiter` *arb)
 - Get the number of contact points for this arbiter.*
- `cpVect cpArbiterGetNormal` (const `cpArbiter` *arb)
 - Get the normal of the collision.*
- `cpVect cpArbiterGetPointA` (const `cpArbiter` *arb, int i)
 - Get the position of the *i*th contact point on the surface of the first shape.*
- `cpVect cpArbiterGetPointB` (const `cpArbiter` *arb, int i)
 - Get the position of the *i*th contact point on the surface of the second shape.*
- `cpFloat cpArbiterGetDepth` (const `cpArbiter` *arb, int i)
 - Get the depth of the *i*th contact point.*
- `cpBool cpArbiterCallWildcardBeginA` (`cpArbiter` *arb, `cpSpace` *space)
 - If you want a custom callback to invoke the wildcard callback for the first collision type, you must call this function explicitly.*
- `cpBool cpArbiterCallWildcardBeginB` (`cpArbiter` *arb, `cpSpace` *space)
 - If you want a custom callback to invoke the wildcard callback for the second collision type, you must call this function explicitly.*
- `cpBool cpArbiterCallWildcardPreSolveA` (`cpArbiter` *arb, `cpSpace` *space)
 - If you want a custom callback to invoke the wildcard callback for the first collision type, you must call this function explicitly.*
- `cpBool cpArbiterCallWildcardPreSolveB` (`cpArbiter` *arb, `cpSpace` *space)
 - If you want a custom callback to invoke the wildcard callback for the second collision type, you must call this function explicitly.*
- `void cpArbiterCallWildcardPostSolveA` (`cpArbiter` *arb, `cpSpace` *space)
 - If you want a custom callback to invoke the wildcard callback for the first collision type, you must call this function explicitly.*
- `void cpArbiterCallWildcardPostSolveB` (`cpArbiter` *arb, `cpSpace` *space)
 - If you want a custom callback to invoke the wildcard callback for the second collision type, you must call this function explicitly.*
- `void cpArbiterCallWildcardSeparateA` (`cpArbiter` *arb, `cpSpace` *space)
 - If you want a custom callback to invoke the wildcard callback for the first collision type, you must call this function explicitly.*
- `void cpArbiterCallWildcardSeparateB` (`cpArbiter` *arb, `cpSpace` *space)
 - If you want a custom callback to invoke the wildcard callback for the second collision type, you must call this function explicitly.*

3.4.1 Detailed Description

The `cpArbiter` struct tracks pairs of colliding shapes.

They are also used in conjunction with collision handler callbacks allowing you to retrieve information on the collision or change it. A unique arbiter value is used for each pair of colliding objects. It persists until the shapes separate.

3.4.2 Function Documentation

3.4.2.1 `cpBool cpArbiterCallWildcardBeginA` (`cpArbiter` * arb, `cpSpace` * space)

If you want a custom callback to invoke the wildcard callback for the first collision type, you must call this function explicitly.

You must decide how to handle the wildcard's return value since it may disagree with the other wildcard handler's return value or your own.

3.4.2.2 cpBool cpArbiterCallWildcardBeginB (cpArbiter * arb, cpSpace * space)

If you want a custom callback to invoke the wildcard callback for the second collision type, you must call this function explicitly.

You must decide how to handle the wildcard's return value since it may disagree with the other wildcard handler's return value or your own.

3.4.2.3 cpBool cpArbiterCallWildcardPreSolveA (cpArbiter * arb, cpSpace * space)

If you want a custom callback to invoke the wildcard callback for the first collision type, you must call this function explicitly.

You must decide how to handle the wildcard's return value since it may disagree with the other wildcard handler's return value or your own.

3.4.2.4 cpBool cpArbiterCallWildcardPreSolveB (cpArbiter * arb, cpSpace * space)

If you want a custom callback to invoke the wildcard callback for the second collision type, you must call this function explicitly.

You must decide how to handle the wildcard's return value since it may disagree with the other wildcard handler's return value or your own.

3.4.2.5 void cpArbiterGetBodies (const cpArbiter * arb, cpBody ** a, cpBody ** b)

Return the colliding bodies involved for this arbiter.

The order of the cpSpace.collision_type the bodies are associated with values will match the order set when the collision handler was registered.

3.4.2.6 void cpArbiterGetShapes (const cpArbiter * arb, cpShape ** a, cpShape ** b)

Return the colliding shapes involved for this arbiter.

The order of their cpSpace.collision_type values will match the order set when the collision handler was registered.

3.4.2.7 cpBool cpArbiterIgnore (cpArbiter * arb)

Mark a collision pair to be ignored until the two objects separate.

Pre-solve and post-solve callbacks will not be called, but the separate callback will be called.

3.4.2.8 void cpArbiterSetContactPointSet (cpArbiter * arb, cpContactPointSet * set)

Replace the contact point set for an arbiter.

This can be a very powerful feature, but use it with caution!

3.4.2.9 void cpArbiterSetUserData (cpArbiter * arb, cpDataPointer userData)

Set a user data point associated with this pair of colliding objects.

If you need to perform any cleanup for this pointer, you must do it yourself, in the separate callback for instance.

3.4.2.10 cpVect cpArbiterTotalImpulse (const cpArbiter * arb)

Calculate the total impulse including the friction that was applied by this arbiter.

This function should only be called from a post-solve, post-step or cpBodyEachArbiter callback.

3.4.2.11 cpFloat cpArbiterTotalKE (const cpArbiter * arb)

Calculate the amount of energy lost in a collision including static, but not dynamic friction.

This function should only be called from a post-solve, post-step or cpBodyEachArbiter callback.

3.5 cpBB

Chipmunk's axis-aligned 2D bounding box type along with a few handy routines.

Classes

- struct `cpBB`
Chipmunk's axis-aligned 2D bounding box type. (left, bottom, right, top)

Typedefs

- typedef struct `cpBB cpBB`
Chipmunk's axis-aligned 2D bounding box type. (left, bottom, right, top)

Functions

- static `cpBB cpBBNew` (const `cpFloat` l, const `cpFloat` b, const `cpFloat` r, const `cpFloat` t)
Convenience constructor for `cpBB` structs.
- static `cpBB cpBBNewForExtents` (const `cpVect` c, const `cpFloat` hw, const `cpFloat` hh)
Constructs a `cpBB` centered on a point with the given extents (half sizes).
- static `cpBB cpBBNewForCircle` (const `cpVect` p, const `cpFloat` r)
Constructs a `cpBB` for a circle with the given position and radius.
- static `cpBool cpBBIntersects` (const `cpBB` a, const `cpBB` b)
Returns true if `a` and `b` intersect.
- static `cpBool cpBBContainsBB` (const `cpBB` bb, const `cpBB` other)
Returns true if `other` lies completely within `bb`.
- static `cpBool cpBBContainsVect` (const `cpBB` bb, const `cpVect` v)
Returns true if `bb` contains `v`.
- static `cpBB cpBBMerge` (const `cpBB` a, const `cpBB` b)
Returns a bounding box that holds both bounding boxes.
- static `cpBB cpBBExpand` (const `cpBB` bb, const `cpVect` v)
Returns a bounding box that holds both `bb` and `v`.
- static `cpVect cpBBCenter` (`cpBB` bb)
Returns the center of a bounding box.
- static `cpFloat cpBBArea` (`cpBB` bb)
Returns the area of the bounding box.
- static `cpFloat cpBBMergedArea` (`cpBB` a, `cpBB` b)
Merges `a` and `b` and returns the area of the merged bounding box.
- static `cpFloat cpBBSegmentQuery` (`cpBB` bb, `cpVect` a, `cpVect` b)
Returns the fraction along the segment query the `cpBB` is hit. Returns `INFINITY` if it doesn't hit.
- static `cpBool cpBBIntersectsSegment` (`cpBB` bb, `cpVect` a, `cpVect` b)
Return true if the bounding box intersects the line segment with ends `a` and `b`.
- static `cpVect cpBBClampVect` (const `cpBB` bb, const `cpVect` v)
Clamp a vector to a bounding box.
- static `cpVect cpBBWrapVect` (const `cpBB` bb, const `cpVect` v)
Wrap a vector to a bounding box.
- static `cpBB cpBBOffset` (const `cpBB` bb, const `cpVect` v)
Returns a bounding box offseted by `v`.

3.5.1 Detailed Description

Chipmunk's axis-aligned 2D bounding box type along with a few handy routines.

3.6 cpBody

Chipmunk's rigid body type.

Typedefs

- typedef enum [cpBodyType](#) **cpBodyType**
- typedef void(* [cpBodyVelocityFunc](#)) ([cpBody](#) *body, [cpVect](#) gravity, [cpFloat](#) damping, [cpFloat](#) dt)
Rigid body velocity update function type.
- typedef void(* [cpBodyPositionFunc](#)) ([cpBody](#) *body, [cpFloat](#) dt)
Rigid body position update function type.
- typedef void(* [cpBodyShapeIteratorFunc](#)) ([cpBody](#) *body, [cpShape](#) *shape, void *data)
Body/shape iterator callback function type.
- typedef void(* [cpBodyConstraintIteratorFunc](#)) ([cpBody](#) *body, [cpConstraint](#) *constraint, void *data)
Body/constraint iterator callback function type.
- typedef void(* [cpBodyArbiterIteratorFunc](#)) ([cpBody](#) *body, [cpArbiter](#) *arbiter, void *data)
Body/arbiter iterator callback function type.

Enumerations

- enum [cpBodyType](#) { [CP_BODY_TYPE_DYNAMIC](#), [CP_BODY_TYPE_KINEMATIC](#), [CP_BODY_TYPE_STATIC](#) }

Functions

- [cpBody](#) * [cpBodyAlloc](#) (void)
Allocate a [cpBody](#).
- [cpBody](#) * [cpBodyInit](#) ([cpBody](#) *body, [cpFloat](#) mass, [cpFloat](#) moment)
Initialize a [cpBody](#).
- [cpBody](#) * [cpBodyNew](#) ([cpFloat](#) mass, [cpFloat](#) moment)
Allocate and initialize a [cpBody](#).
- [cpBody](#) * [cpBodyNewKinematic](#) (void)
Allocate and initialize a [cpBody](#), and set it as a kinematic body.
- [cpBody](#) * [cpBodyNewStatic](#) (void)
Allocate and initialize a [cpBody](#), and set it as a static body.
- void [cpBodyDestroy](#) ([cpBody](#) *body)
Destroy a [cpBody](#).
- void [cpBodyFree](#) ([cpBody](#) *body)
Destroy and free a [cpBody](#).
- void [cpBodyActivate](#) ([cpBody](#) *body)
Wake up a sleeping or idle body.
- void [cpBodyActivateStatic](#) ([cpBody](#) *body, [cpShape](#) *filter)
Wake up any sleeping or idle bodies touching a static body.
- void [cpBodySleep](#) ([cpBody](#) *body)
Force a body to fall asleep immediately.
- void [cpBodySleepWithGroup](#) ([cpBody](#) *body, [cpBody](#) *group)
Force a body to fall asleep immediately along with other bodies in a group.
- [cpBool](#) [cpBodyIsSleeping](#) (const [cpBody](#) *body)
Returns true if the body is sleeping.
- [cpBodyType](#) [cpBodyGetType](#) ([cpBody](#) *body)

- Get the type of the body.*

 - void `cpBodySetType` (`cpBody` *body, `cpBodyType` type)

Set the type of the body.
- `cpSpace` * `cpBodyGetSpace` (const `cpBody` *body)

Get the space this body is added to.
- `cpFloat` `cpBodyGetMass` (const `cpBody` *body)

Get the mass of the body.
- void `cpBodySetMass` (`cpBody` *body, `cpFloat` m)

Set the mass of the body.
- `cpFloat` `cpBodyGetMoment` (const `cpBody` *body)

Get the moment of inertia of the body.
- void `cpBodySetMoment` (`cpBody` *body, `cpFloat` i)

Set the moment of inertia of the body.
- `cpVect` `cpBodyGetPosition` (const `cpBody` *body)

Get the position of a body.
- void `cpBodySetPosition` (`cpBody` *body, `cpVect` pos)

Set the position of the body.
- `cpVect` `cpBodyGetCenterOfGravity` (const `cpBody` *body)

Get the offset of the center of gravity in body local coordinates.
- void `cpBodySetCenterOfGravity` (`cpBody` *body, `cpVect` cog)

Set the offset of the center of gravity in body local coordinates.
- `cpVect` `cpBodyGetVelocity` (const `cpBody` *body)

Get the velocity of the body.
- void `cpBodySetVelocity` (`cpBody` *body, `cpVect` velocity)

Set the velocity of the body.
- `cpVect` `cpBodyGetForce` (const `cpBody` *body)

Get the force applied to the body for the next time step.
- void `cpBodySetForce` (`cpBody` *body, `cpVect` force)

Set the force applied to the body for the next time step.
- `cpFloat` `cpBodyGetAngle` (const `cpBody` *body)

Get the angle of the body.
- void `cpBodySetAngle` (`cpBody` *body, `cpFloat` a)

Set the angle of a body.
- `cpFloat` `cpBodyGetAngularVelocity` (const `cpBody` *body)

Get the angular velocity of the body.
- void `cpBodySetAngularVelocity` (`cpBody` *body, `cpFloat` angularVelocity)

Set the angular velocity of the body.
- `cpFloat` `cpBodyGetTorque` (const `cpBody` *body)

Get the torque applied to the body for the next time step.
- void `cpBodySetTorque` (`cpBody` *body, `cpFloat` torque)

Set the torque applied to the body for the next time step.
- `cpVect` `cpBodyGetRotation` (const `cpBody` *body)

Get the rotation vector of the body. (The x basis vector of it's transform.)
- `cpDataPointer` `cpBodyGetUserData` (const `cpBody` *body)

Get the user data pointer assigned to the body.
- void `cpBodySetUserData` (`cpBody` *body, `cpDataPointer` userData)

Set the user data pointer assigned to the body.
- void `cpBodySetVelocityUpdateFunc` (`cpBody` *body, `cpBodyVelocityFunc` velocityFunc)

Set the callback used to update a body's velocity.
- void `cpBodySetPositionUpdateFunc` (`cpBody` *body, `cpBodyPositionFunc` positionFunc)

Set the callback used to update a body's position.

- void `cpBodyUpdateVelocity` (`cpBody` *body, `cpVect` gravity, `cpFloat` damping, `cpFloat` dt)
Default velocity integration function.
- void `cpBodyUpdatePosition` (`cpBody` *body, `cpFloat` dt)
Default position integration function.
- `cpVect` `cpBodyLocalToWorld` (const `cpBody` *body, const `cpVect` point)
Convert body relative/local coordinates to absolute/world coordinates.
- `cpVect` `cpBodyWorldToLocal` (const `cpBody` *body, const `cpVect` point)
Convert body absolute/world coordinates to relative/local coordinates.
- void `cpBodyApplyForceAtWorldPoint` (`cpBody` *body, `cpVect` force, `cpVect` point)
Apply a force to a body. Both the force and point are expressed in world coordinates.
- void `cpBodyApplyForceAtLocalPoint` (`cpBody` *body, `cpVect` force, `cpVect` point)
Apply a force to a body. Both the force and point are expressed in body local coordinates.
- void `cpBodyApplyImpulseAtWorldPoint` (`cpBody` *body, `cpVect` impulse, `cpVect` point)
Apply an impulse to a body. Both the impulse and point are expressed in world coordinates.
- void `cpBodyApplyImpulseAtLocalPoint` (`cpBody` *body, `cpVect` impulse, `cpVect` point)
Apply an impulse to a body. Both the impulse and point are expressed in body local coordinates.
- `cpVect` `cpBodyGetVelocityAtWorldPoint` (const `cpBody` *body, `cpVect` point)
Get the velocity on a body (in world units) at a point on the body in world coordinates.
- `cpVect` `cpBodyGetVelocityAtLocalPoint` (const `cpBody` *body, `cpVect` point)
Get the velocity on a body (in world units) at a point on the body in local coordinates.
- `cpFloat` `cpBodyKineticEnergy` (const `cpBody` *body)
Get the amount of kinetic energy contained by the body.
- void `cpBodyEachShape` (`cpBody` *body, `cpBodyShapeIteratorFunc` func, void *data)
Call func once for each shape attached to body and added to the space.
- void `cpBodyEachConstraint` (`cpBody` *body, `cpBodyConstraintIteratorFunc` func, void *data)
Call func once for each constraint attached to body and added to the space.
- void `cpBodyEachArbiter` (`cpBody` *body, `cpBodyArbiterIteratorFunc` func, void *data)
Call func once for each arbiter that is currently active on the body.

3.6.1 Detailed Description

Chipmunk's rigid body type.

Rigid bodies hold the physical properties of an object like it's mass, and position and velocity of it's center of gravity. They don't have a shape on their own. They are given a shape by creating collision shapes (`cpShape`) that point to the body.

3.6.2 Enumeration Type Documentation

3.6.2.1 enum `cpBodyType`

Enumerator

`CP_BODY_TYPE_DYNAMIC` A dynamic body is one that is affected by gravity, forces, and collisions. This is the default body type.

`CP_BODY_TYPE_KINEMATIC` A kinematic body is an infinite mass, user controlled body that is not affected by gravity, forces or collisions. Instead the body only moves based on it's velocity. Dynamic bodies collide normally with kinematic bodies, though the kinematic body will be unaffected. Collisions between two kinematic bodies, or a kinematic body and a static body produce collision callbacks, but no collision response.

`CP_BODY_TYPE_STATIC` A static body is a body that never (or rarely) moves. If you move a static body, you must call one of the `cpSpaceReindex*()` functions. Chipmunk uses this information to optimize the collision detection. Static bodies do not produce collision callbacks when colliding with other static bodies.

3.6.3 Function Documentation

3.6.3.1 void cpBodySetPositionUpdateFunc (cpBody * *body*, cpBodyPositionFunc *positionFunc*)

Set the callback used to update a body's position.

NOTE: It's not generally recommended to override this unless you call the default position update function.

3.7 cpConstraint

Constraints connect two [cpBody](#) objects together.

Typedefs

- typedef void(* [cpConstraintPreSolveFunc](#)) ([cpConstraint](#) *constraint, [cpSpace](#) *space)
Callback function type that gets called before solving a joint.
- typedef void(* [cpConstraintPostSolveFunc](#)) ([cpConstraint](#) *constraint, [cpSpace](#) *space)
Callback function type that gets called after solving a joint.

Functions

- void [cpConstraintDestroy](#) ([cpConstraint](#) *constraint)
Destroy a constraint.
- void [cpConstraintFree](#) ([cpConstraint](#) *constraint)
Destroy and free a constraint.
- [cpSpace](#) * [cpConstraintGetSpace](#) (const [cpConstraint](#) *constraint)
Get the [cpSpace](#) this constraint is added to.
- [cpBody](#) * [cpConstraintGetBodyA](#) (const [cpConstraint](#) *constraint)
Get the first body the constraint is attached to.
- [cpBody](#) * [cpConstraintGetBodyB](#) (const [cpConstraint](#) *constraint)
Get the second body the constraint is attached to.
- [cpFloat](#) [cpConstraintGetMaxForce](#) (const [cpConstraint](#) *constraint)
Get the maximum force that this constraint is allowed to use.
- void [cpConstraintSetMaxForce](#) ([cpConstraint](#) *constraint, [cpFloat](#) maxForce)
Set the maximum force that this constraint is allowed to use. (defaults to INFINITY)
- [cpFloat](#) [cpConstraintGetErrorBias](#) (const [cpConstraint](#) *constraint)
Get rate at which joint error is corrected.
- void [cpConstraintSetErrorBias](#) ([cpConstraint](#) *constraint, [cpFloat](#) errorBias)
Set rate at which joint error is corrected.
- [cpFloat](#) [cpConstraintGetMaxBias](#) (const [cpConstraint](#) *constraint)
Get the maximum rate at which joint error is corrected.
- void [cpConstraintSetMaxBias](#) ([cpConstraint](#) *constraint, [cpFloat](#) maxBias)
Set the maximum rate at which joint error is corrected. (defaults to INFINITY)
- [cpBool](#) [cpConstraintGetCollideBodies](#) (const [cpConstraint](#) *constraint)
Get if the two bodies connected by the constraint are allowed to collide or not.
- void [cpConstraintSetCollideBodies](#) ([cpConstraint](#) *constraint, [cpBool](#) collideBodies)
Set if the two bodies connected by the constraint are allowed to collide or not. (defaults to cpFalse)
- [cpConstraintPreSolveFunc](#) [cpConstraintGetPreSolveFunc](#) (const [cpConstraint](#) *constraint)
Get the pre-solve function that is called before the solver runs.
- void [cpConstraintSetPreSolveFunc](#) ([cpConstraint](#) *constraint, [cpConstraintPreSolveFunc](#) preSolveFunc)
Set the pre-solve function that is called before the solver runs.
- [cpConstraintPostSolveFunc](#) [cpConstraintGetPostSolveFunc](#) (const [cpConstraint](#) *constraint)
Get the post-solve function that is called before the solver runs.
- void [cpConstraintSetPostSolveFunc](#) ([cpConstraint](#) *constraint, [cpConstraintPostSolveFunc](#) postSolveFunc)
Set the post-solve function that is called before the solver runs.
- [cpDataPointer](#) [cpConstraintGetUserData](#) (const [cpConstraint](#) *constraint)
Get the user definable data pointer for this constraint.
- void [cpConstraintSetUserData](#) ([cpConstraint](#) *constraint, [cpDataPointer](#) userData)
Set the user definable data pointer for this constraint.
- [cpFloat](#) [cpConstraintGetImpulse](#) ([cpConstraint](#) *constraint)
Get the last impulse applied by this constraint.

3.7.1 Detailed Description

Constraints connect two [cpBody](#) objects together.

[cpConstraint](#) is the base constraint struct that the other constraints build off of.

3.7.2 Function Documentation

3.7.2.1 void cpConstraintSetErrorBias (cpConstraint * *constraint*, cpFloat *errorBias*)

Set rate at which joint error is corrected.

Defaults to $\text{pow}(1.0 - 0.1, 60.0)$ meaning that it will correct 10% of the error every 1/60th of a second.

3.8 cpDampedRotarySpring

Like a [cpDampedSpring](#), but operates in a rotational fashion.

Typedefs

- typedef [cpFloat](#)(* [cpDampedRotarySpringTorqueFunc](#)) (struct [cpConstraint](#) *spring, [cpFloat](#) relativeAngle)
Function type used for damped rotary spring force callbacks.

Functions

- [cpBool](#) [cpConstraintIsDampedRotarySpring](#) (const [cpConstraint](#) *constraint)
Check if a constraint is a damped rotary springs.
- [cpDampedRotarySpring](#) * [cpDampedRotarySpringAlloc](#) (void)
Allocate a damped rotary spring.
- [cpDampedRotarySpring](#) * [cpDampedRotarySpringInit](#) ([cpDampedRotarySpring](#) *joint, [cpBody](#) *a, [cpBody](#) *b, [cpFloat](#) restAngle, [cpFloat](#) stiffness, [cpFloat](#) damping)
Initialize a damped rotary spring.
- [cpConstraint](#) * [cpDampedRotarySpringNew](#) ([cpBody](#) *a, [cpBody](#) *b, [cpFloat](#) restAngle, [cpFloat](#) stiffness, [cpFloat](#) damping)
Allocate and initialize a damped rotary spring.
- [cpFloat](#) [cpDampedRotarySpringGetRestAngle](#) (const [cpConstraint](#) *constraint)
Get the rest length of the spring.
- void [cpDampedRotarySpringSetRestAngle](#) ([cpConstraint](#) *constraint, [cpFloat](#) restAngle)
Set the rest length of the spring.
- [cpFloat](#) [cpDampedRotarySpringGetStiffness](#) (const [cpConstraint](#) *constraint)
Get the stiffness of the spring in force/distance.
- void [cpDampedRotarySpringSetStiffness](#) ([cpConstraint](#) *constraint, [cpFloat](#) stiffness)
Set the stiffness of the spring in force/distance.
- [cpFloat](#) [cpDampedRotarySpringGetDamping](#) (const [cpConstraint](#) *constraint)
Get the damping of the spring.
- void [cpDampedRotarySpringSetDamping](#) ([cpConstraint](#) *constraint, [cpFloat](#) damping)
Set the damping of the spring.
- [cpDampedRotarySpringTorqueFunc](#) [cpDampedRotarySpringGetSpringTorqueFunc](#) (const [cpConstraint](#) *constraint)
Get the damped rotary spring force callback.
- void [cpDampedRotarySpringSetSpringTorqueFunc](#) ([cpConstraint](#) *constraint, [cpDampedRotarySpringTorqueFunc](#) springTorqueFunc)
Set the damped rotary spring force callback.

3.8.1 Detailed Description

Like a [cpDampedSpring](#), but operates in a rotational fashion.

3.9 cpDampedSpring

A spring with a damper.

Typedefs

- typedef `cpFloat`(* `cpDampedSpringForceFunc`) (`cpConstraint` *spring, `cpFloat` dist)
Function type used for damped spring force callbacks.

Functions

- `cpBool` `cpConstraintIsDampedSpring` (`const cpConstraint` *constraint)
Check if a constraint is a damped spring.
- `cpDampedSpring` * `cpDampedSpringAlloc` (`void`)
Allocate a damped spring.
- `cpDampedSpring` * `cpDampedSpringInit` (`cpDampedSpring` *joint, `cpBody` *a, `cpBody` *b, `cpVect` anchorA, `cpVect` anchorB, `cpFloat` restLength, `cpFloat` stiffness, `cpFloat` damping)
Initialize a damped spring.
- `cpConstraint` * `cpDampedSpringNew` (`cpBody` *a, `cpBody` *b, `cpVect` anchorA, `cpVect` anchorB, `cpFloat` restLength, `cpFloat` stiffness, `cpFloat` damping)
Allocate and initialize a damped spring.
- `cpVect` `cpDampedSpringGetAnchorA` (`const cpConstraint` *constraint)
Get the location of the first anchor relative to the first body.
- `void` `cpDampedSpringSetAnchorA` (`cpConstraint` *constraint, `cpVect` anchorA)
Set the location of the first anchor relative to the first body.
- `cpVect` `cpDampedSpringGetAnchorB` (`const cpConstraint` *constraint)
Get the location of the second anchor relative to the second body.
- `void` `cpDampedSpringSetAnchorB` (`cpConstraint` *constraint, `cpVect` anchorB)
Set the location of the second anchor relative to the second body.
- `cpFloat` `cpDampedSpringGetRestLength` (`const cpConstraint` *constraint)
Get the rest length of the spring.
- `void` `cpDampedSpringSetRestLength` (`cpConstraint` *constraint, `cpFloat` restLength)
Set the rest length of the spring.
- `cpFloat` `cpDampedSpringGetStiffness` (`const cpConstraint` *constraint)
Get the stiffness of the spring in force/distance.
- `void` `cpDampedSpringSetStiffness` (`cpConstraint` *constraint, `cpFloat` stiffness)
Set the stiffness of the spring in force/distance.
- `cpFloat` `cpDampedSpringGetDamping` (`const cpConstraint` *constraint)
Get the damping of the spring.
- `void` `cpDampedSpringSetDamping` (`cpConstraint` *constraint, `cpFloat` damping)
Set the damping of the spring.
- `cpDampedSpringForceFunc` `cpDampedSpringGetSpringForceFunc` (`const cpConstraint` *constraint)
Get the damped spring force callback.
- `void` `cpDampedSpringSetSpringForceFunc` (`cpConstraint` *constraint, `cpDampedSpringForceFunc` spring↔
ForceFunc)
Set the damped spring force callback.

3.9.1 Detailed Description

A spring with a damper.

While a spring is not technically a constraint, the damper is. The spring forces are simply a convenience.

3.10 cpGearJoint

Maintains a specific angular velocity between the two bodies.

Functions

- `cpBool cpConstraintIsGearJoint` (const `cpConstraint` *constraint)
Check if a constraint is a gear joint.
- `cpGearJoint * cpGearJointAlloc` (void)
Allocate a gear joint.
- `cpGearJoint * cpGearJointInit` (`cpGearJoint` *joint, `cpBody` *a, `cpBody` *b, `cpFloat` phase, `cpFloat` ratio)
Initialize a gear joint.
- `cpConstraint * cpGearJointNew` (`cpBody` *a, `cpBody` *b, `cpFloat` phase, `cpFloat` ratio)
Allocate and initialize a gear joint.
- `cpFloat cpGearJointGetPhase` (const `cpConstraint` *constraint)
Get the phase offset of the gears.
- void `cpGearJointSetPhase` (`cpConstraint` *constraint, `cpFloat` phase)
Set the phase offset of the gears.
- `cpFloat cpGearJointGetRatio` (const `cpConstraint` *constraint)
Get the ratio of a gear joint.
- void `cpGearJointSetRatio` (`cpConstraint` *constraint, `cpFloat` ratio)
Set the ratio of a gear joint.

3.10.1 Detailed Description

Maintains a specific angular velocity between the two bodies.

3.11 cpGrooveJoint

Similar to a pivot joint, but one of the anchors is a line segment that the pivot can slide in.

Functions

- `cpBool cpConstraintIsGrooveJoint` (const `cpConstraint` *constraint)
Check if a constraint is a groove joint.
- `cpGrooveJoint * cpGrooveJointAlloc` (void)
Allocate a groove joint.
- `cpGrooveJoint * cpGrooveJointInit` (cpGrooveJoint *joint, cpBody *a, cpBody *b, cpVect groove_a, cpVect groove_b, cpVect anchorB)
Initialize a groove joint.
- `cpConstraint * cpGrooveJointNew` (cpBody *a, cpBody *b, cpVect groove_a, cpVect groove_b, cpVect anchorB)
Allocate and initialize a groove joint.
- `cpVect cpGrooveJointGetGrooveA` (const `cpConstraint` *constraint)
Get the first endpoint of the groove relative to the first body.
- `void cpGrooveJointSetGrooveA` (cpConstraint *constraint, cpVect grooveA)
Set the first endpoint of the groove relative to the first body.
- `cpVect cpGrooveJointGetGrooveB` (const `cpConstraint` *constraint)
Get the first endpoint of the groove relative to the first body.
- `void cpGrooveJointSetGrooveB` (cpConstraint *constraint, cpVect grooveB)
Set the first endpoint of the groove relative to the first body.
- `cpVect cpGrooveJointGetAnchorB` (const `cpConstraint` *constraint)
Get the location of the second anchor relative to the second body.
- `void cpGrooveJointSetAnchorB` (cpConstraint *constraint, cpVect anchorB)
Set the location of the second anchor relative to the second body.

3.11.1 Detailed Description

Similar to a pivot joint, but one of the anchors is a line segment that the pivot can slide in.

3.12 cpPinJoint

The two anchor points are always the same distance apart.

Functions

- `cpBool cpConstraintIsPinJoint` (const `cpConstraint` *constraint)
Check if a constraint is a pin joint.
- `cpPinJoint * cpPinJointAlloc` (void)
Allocate a pin joint.
- `cpPinJoint * cpPinJointInit` (cpPinJoint *joint, cpBody *a, cpBody *b, cpVect anchorA, cpVect anchorB)
Initialize a pin joint.
- `cpConstraint * cpPinJointNew` (cpBody *a, cpBody *b, cpVect anchorA, cpVect anchorB)
Allocate and initialize a pin joint.
- `cpVect cpPinJointGetAnchorA` (const `cpConstraint` *constraint)
Get the location of the first anchor relative to the first body.
- void `cpPinJointSetAnchorA` (cpConstraint *constraint, cpVect anchorA)
Set the location of the first anchor relative to the first body.
- `cpVect cpPinJointGetAnchorB` (const `cpConstraint` *constraint)
Get the location of the second anchor relative to the second body.
- void `cpPinJointSetAnchorB` (cpConstraint *constraint, cpVect anchorB)
Set the location of the second anchor relative to the second body.
- `cpFloat cpPinJointGetDist` (const `cpConstraint` *constraint)
Get the distance the joint will maintain between the two anchors.
- void `cpPinJointSetDist` (cpConstraint *constraint, cpFloat dist)
Set the distance the joint will maintain between the two anchors.

3.12.1 Detailed Description

The two anchor points are always the same distance apart.

3.13 cpPivotJoint

Pivot joints hold two points on two bodies together allowing them to rotate freely around the pivot.

Functions

- `cpBool cpConstraintIsPivotJoint` (const `cpConstraint` *constraint)
Check if a constraint is a pivot joint.
- `cpPivotJoint * cpPivotJointAlloc` (void)
Allocate a pivot joint.
- `cpPivotJoint * cpPivotJointInit` (cpPivotJoint *joint, cpBody *a, cpBody *b, cpVect anchorA, cpVect anchorB)
Initialize a pivot joint.
- `cpConstraint * cpPivotJointNew` (cpBody *a, cpBody *b, cpVect pivot)
Allocate and initialize a pivot joint.
- `cpConstraint * cpPivotJointNew2` (cpBody *a, cpBody *b, cpVect anchorA, cpVect anchorB)
Allocate and initialize a pivot joint with specific anchors.
- `cpVect cpPivotJointGetAnchorA` (const `cpConstraint` *constraint)
Get the location of the first anchor relative to the first body.
- void `cpPivotJointSetAnchorA` (cpConstraint *constraint, cpVect anchorA)
Set the location of the first anchor relative to the first body.
- `cpVect cpPivotJointGetAnchorB` (const `cpConstraint` *constraint)
Get the location of the second anchor relative to the second body.
- void `cpPivotJointSetAnchorB` (cpConstraint *constraint, cpVect anchorB)
Set the location of the second anchor relative to the second body.

3.13.1 Detailed Description

Pivot joints hold two points on two bodies together allowing them to rotate freely around the pivot.

3.14 cpPolyShape

A convex polygon shape.

Functions

- `cpPolyShape * cpPolyShapeAlloc` (void)
Allocate a polygon shape.
- `cpPolyShape * cpPolyShapeInit` (cpPolyShape *poly, cpBody *body, int count, const cpVect *verts, cpTransform transform, cpFloat radius)
Initialize a polygon shape with rounded corners.
- `cpPolyShape * cpPolyShapeInitRaw` (cpPolyShape *poly, cpBody *body, int count, const cpVect *verts, cpFloat radius)
Initialize a polygon shape with rounded corners.
- `cpShape * cpPolyShapeNew` (cpBody *body, int count, const cpVect *verts, cpTransform transform, cpFloat radius)
Allocate and initialize a polygon shape with rounded corners.
- `cpShape * cpPolyShapeNewRaw` (cpBody *body, int count, const cpVect *verts, cpFloat radius)
Allocate and initialize a polygon shape with rounded corners.
- `cpPolyShape * cpBoxShapeInit` (cpPolyShape *poly, cpBody *body, cpFloat width, cpFloat height, cpFloat radius)
Initialize a box shaped polygon shape with rounded corners.
- `cpPolyShape * cpBoxShapeInit2` (cpPolyShape *poly, cpBody *body, cpBB box, cpFloat radius)
Initialize an offset box shaped polygon shape with rounded corners.
- `cpShape * cpBoxShapeNew` (cpBody *body, cpFloat width, cpFloat height, cpFloat radius)
Allocate and initialize a box shaped polygon shape.
- `cpShape * cpBoxShapeNew2` (cpBody *body, cpBB box, cpFloat radius)
Allocate and initialize an offset box shaped polygon shape.
- `int cpPolyShapeGetCount` (const cpShape *shape)
Get the number of verts in a polygon shape.
- `cpVect cpPolyShapeGetVert` (const cpShape *shape, int index)
*Get the *i*th vertex of a polygon shape.*
- `cpFloat cpPolyShapeGetRadius` (const cpShape *shape)
Get the radius of a polygon shape.

3.14.1 Detailed Description

A convex polygon shape.

Slowest, but most flexible collision shape.

3.14.2 Function Documentation

3.14.2.1 cpPolyShape* cpPolyShapeInit (cpPolyShape * poly, cpBody * body, int count, const cpVect * verts, cpTransform transform, cpFloat radius)

Initialize a polygon shape with rounded corners.

A convex hull will be created from the vertexes.

3.14.2.2 `cpPolyShape* cpPolyShapeInitRaw (cpPolyShape * poly, cpBody * body, int count, const cpVect * verts, cpFloat radius)`

Initialize a polygon shape with rounded corners.

The vertexes must be convex with a counter-clockwise winding.

3.14.2.3 `cpShape* cpPolyShapeNew (cpBody * body, int count, const cpVect * verts, cpTransform transform, cpFloat radius)`

Allocate and initialize a polygon shape with rounded corners.

A convex hull will be created from the vertexes.

3.14.2.4 `cpShape* cpPolyShapeNewRaw (cpBody * body, int count, const cpVect * verts, cpFloat radius)`

Allocate and initialize a polygon shape with rounded corners.

The vertexes must be convex with a counter-clockwise winding.

3.15 cpRatchetJoint

Create rotary ratches similar to a socket wrench.

Functions

- `cpBool cpConstraintIsRatchetJoint` (const `cpConstraint` *constraint)
Check if a constraint is a ratchet joint.
- `cpRatchetJoint * cpRatchetJointAlloc` (void)
Allocate a ratchet joint.
- `cpRatchetJoint * cpRatchetJointInit` (`cpRatchetJoint` *joint, `cpBody` *a, `cpBody` *b, `cpFloat` phase, `cpFloat` ratchet)
Initialize a ratched joint.
- `cpConstraint * cpRatchetJointNew` (`cpBody` *a, `cpBody` *b, `cpFloat` phase, `cpFloat` ratchet)
Allocate and initialize a ratchet joint.
- `cpFloat cpRatchetJointGetAngle` (const `cpConstraint` *constraint)
Get the angle of the current ratchet tooth.
- void `cpRatchetJointSetAngle` (`cpConstraint` *constraint, `cpFloat` angle)
Set the angle of the current ratchet tooth.
- `cpFloat cpRatchetJointGetPhase` (const `cpConstraint` *constraint)
Get the phase offset of the ratchet.
- void `cpRatchetJointSetPhase` (`cpConstraint` *constraint, `cpFloat` phase)
Set the phase offset of the ratchet.
- `cpFloat cpRatchetJointGetRatchet` (const `cpConstraint` *constraint)
Get the angular distance of each ratchet.
- void `cpRatchetJointSetRatchet` (`cpConstraint` *constraint, `cpFloat` ratchet)
Set the angular distance of each ratchet.

3.15.1 Detailed Description

Create rotary ratches similar to a socket wrench.

Forces one body to only follow one direction of rotation from the other body

3.16 cpRotaryLimitJoint

Constrains the bodies' orientations to be within a certain angle of each other.

Functions

- `cpBool cpConstraintIsRotaryLimitJoint` (const `cpConstraint` *constraint)
Check if a constraint is a rotary limit joint.
- `cpRotaryLimitJoint * cpRotaryLimitJointAlloc` (void)
Allocate a damped rotary limit joint.
- `cpRotaryLimitJoint * cpRotaryLimitJointInit` (cpRotaryLimitJoint *joint, cpBody *a, cpBody *b, cpFloat min, cpFloat max)
Initialize a damped rotary limit joint.
- `cpConstraint * cpRotaryLimitJointNew` (cpBody *a, cpBody *b, cpFloat min, cpFloat max)
Allocate and initialize a damped rotary limit joint.
- `cpFloat cpRotaryLimitJointGetMin` (const `cpConstraint` *constraint)
Get the minimum distance the joint will maintain between the two anchors.
- `void cpRotaryLimitJointSetMin` (cpConstraint *constraint, cpFloat min)
Set the minimum distance the joint will maintain between the two anchors.
- `cpFloat cpRotaryLimitJointGetMax` (const `cpConstraint` *constraint)
Get the maximum distance the joint will maintain between the two anchors.
- `void cpRotaryLimitJointSetMax` (cpConstraint *constraint, cpFloat max)
Set the maximum distance the joint will maintain between the two anchors.

3.16.1 Detailed Description

Constrains the bodies' orientations to be within a certain angle of each other.

3.17 cpShape

The `cpShape` struct defines the shape of a rigid body.

Classes

- struct `cpPointQueryInfo`
Point query info struct.
- struct `cpSegmentQueryInfo`
Segment query info struct.
- struct `cpShapeFilter`
Fast collision filtering type that is used to determine if two objects collide before calling collision or query callbacks.

Typedefs

- typedef struct `cpPointQueryInfo` `cpPointQueryInfo`
Point query info struct.
- typedef struct `cpSegmentQueryInfo` `cpSegmentQueryInfo`
Segment query info struct.
- typedef struct `cpShapeFilter` `cpShapeFilter`
Fast collision filtering type that is used to determine if two objects collide before calling collision or query callbacks.

Functions

- static `cpShapeFilter` `cpShapeFilterNew` (`cpGroup` group, `cpBitmask` categories, `cpBitmask` mask)
Create a new collision filter.
- void `cpShapeDestroy` (`cpShape` *shape)
Destroy a shape.
- void `cpShapeFree` (`cpShape` *shape)
Destroy and Free a shape.
- `cpBB` `cpShapeCacheBB` (`cpShape` *shape)
Update, cache and return the bounding box of a shape based on the body it's attached to.
- `cpBB` `cpShapeUpdate` (`cpShape` *shape, `cpTransform` transform)
Update, cache and return the bounding box of a shape with an explicit transformation.
- `cpFloat` `cpShapePointQuery` (const `cpShape` *shape, `cpVect` p, `cpPointQueryInfo` *out)
Perform a nearest point query.
- `cpBool` `cpShapeSegmentQuery` (const `cpShape` *shape, `cpVect` a, `cpVect` b, `cpFloat` radius, `cpSegmentQueryInfo` *info)
Perform a segment query against a shape. info must be a pointer to a valid cpSegmentQueryInfo structure.
- `cpContactPointSet` `cpShapesCollide` (const `cpShape` *a, const `cpShape` *b)
Return contact information about two shapes.
- `cpSpace` * `cpShapeGetSpace` (const `cpShape` *shape)
The cpSpace this body is added to.
- `cpBody` * `cpShapeGetBody` (const `cpShape` *shape)
The cpBody this shape is connected to.
- void `cpShapeSetBody` (`cpShape` *shape, `cpBody` *body)
Set the cpBody this shape is connected to.
- `cpFloat` `cpShapeGetMass` (`cpShape` *shape)
Get the mass of the shape if you are having Chipmunk calculate mass properties for you.
- void `cpShapeSetMass` (`cpShape` *shape, `cpFloat` mass)

- Set the mass of this shape to have Chipmunk calculate mass properties for you.*

 - `cpFloat cpShapeGetDensity (cpShape *shape)`

Get the density of the shape if you are having Chipmunk calculate mass properties for you.
- `void cpShapeSetDensity (cpShape *shape, cpFloat density)`

Set the density of this shape to have Chipmunk calculate mass properties for you.
- `cpFloat cpShapeGetMoment (cpShape *shape)`

Get the calculated moment of inertia for this shape.
- `cpFloat cpShapeGetArea (cpShape *shape)`

Get the calculated area of this shape.
- `cpVect cpShapeGetCenterOfGravity (cpShape *shape)`

Get the centroid of this shape.
- `cpBB cpShapeGetBB (const cpShape *shape)`

Get the bounding box that contains the shape given it's current position and angle.
- `cpBool cpShapeGetSensor (const cpShape *shape)`

Get if the shape is set to be a sensor or not.
- `void cpShapeSetSensor (cpShape *shape, cpBool sensor)`

Set if the shape is a sensor or not.
- `cpFloat cpShapeGetElasticity (const cpShape *shape)`

Get the elasticity of this shape.
- `void cpShapeSetElasticity (cpShape *shape, cpFloat elasticity)`

Set the elasticity of this shape.
- `cpFloat cpShapeGetFriction (const cpShape *shape)`

Get the friction of this shape.
- `void cpShapeSetFriction (cpShape *shape, cpFloat friction)`

Set the friction of this shape.
- `cpVect cpShapeGetSurfaceVelocity (const cpShape *shape)`

Get the surface velocity of this shape.
- `void cpShapeSetSurfaceVelocity (cpShape *shape, cpVect surfaceVelocity)`

Set the surface velocity of this shape.
- `cpDataPointer cpShapeGetUserData (const cpShape *shape)`

Get the user definable data pointer of this shape.
- `void cpShapeSetUserData (cpShape *shape, cpDataPointer userData)`

Set the user definable data pointer of this shape.
- `cpCollisionType cpShapeGetCollisionType (const cpShape *shape)`

Set the collision type of this shape.
- `void cpShapeSetCollisionType (cpShape *shape, cpCollisionType collisionType)`

Get the collision type of this shape.
- `cpShapeFilter cpShapeGetFilter (const cpShape *shape)`

Get the collision filtering parameters of this shape.
- `void cpShapeSetFilter (cpShape *shape, cpShapeFilter filter)`

Set the collision filtering parameters of this shape.

Variables

- `static const cpShapeFilter CP_SHAPE_FILTER_ALL = {CP_NO_GROUP, CP_ALL_CATEGORIES, CP_↵ ALL_CATEGORIES}`

Collision filter value for a shape that will collide with anything except CP_SHAPE_FILTER_NONE.
- `static const cpShapeFilter CP_SHAPE_FILTER_NONE = {CP_NO_GROUP, ~CP_ALL_CATEGORIE↵ S, ~CP_ALL_CATEGORIES}`

Collision filter value for a shape that does not collide with anything.

3.17.1 Detailed Description

The `cpShape` struct defines the shape of a rigid body.

`cpShape` is the base struct that the other shapes build off of.

3.17.2 Function Documentation

3.17.2.1 `cpFloat cpShapePointQuery (const cpShape * shape, cpVect p, cpPointQueryInfo * out)`

Perform a nearest point query.

It finds the closest point on the surface of shape to a specific point. The value returned is the distance between the points. A negative distance means the point is inside the shape.

3.17.2.2 `void cpShapeSetBody (cpShape * shape, cpBody * body)`

Set the `cpBody` this shape is connected to.

Can only be used if the shape is not currently added to a space.

3.18 cpCircleShape

A perfect circle shape.

A perfect circle shape.

Fastest and simplest collision shape.

3.19 cpSegmentShape

A beveled (rounded) segment shape.

A beveled (rounded) segment shape.

Meant mainly as a static shape. Can be beveled in order to give them a thickness.

3.20 cpSimpleMotor

Maintains a specific angular relative velocity between two objects.

Typedefs

- typedef struct `cpSimpleMotor` `cpSimpleMotor`
Opaque struct type for simple motors.

Functions

- `cpBool cpConstraintIsSimpleMotor` (const `cpConstraint` *constraint)
Check if a constraint is a simple motor.
- `cpSimpleMotor` * `cpSimpleMotorAlloc` (void)
Allocate a simple motor.
- `cpSimpleMotor` * `cpSimpleMotorInit` (`cpSimpleMotor` *joint, `cpBody` *a, `cpBody` *b, `cpFloat` rate)
initialize a simple motor.
- `cpConstraint` * `cpSimpleMotorNew` (`cpBody` *a, `cpBody` *b, `cpFloat` rate)
Allocate and initialize a simple motor.
- `cpFloat cpSimpleMotorGetRate` (const `cpConstraint` *constraint)
Get the rate of the motor.
- void `cpSimpleMotorSetRate` (`cpConstraint` *constraint, `cpFloat` rate)
Set the rate of the motor.

3.20.1 Detailed Description

Maintains a specific angular relative velocity between two objects.

3.21 cpSlideJoint

Slide joints hold the distance between points on two bodies between a minimum and a maximum.

Functions

- `cpBool cpConstraintIsSlideJoint` (const `cpConstraint` *constraint)
Check if a constraint is a slide joint.
- `cpSlideJoint * cpSlideJointAlloc` (void)
Allocate a slide joint.
- `cpSlideJoint * cpSlideJointInit` (cpSlideJoint *joint, cpBody *a, cpBody *b, cpVect anchorA, cpVect anchorB, cpFloat min, cpFloat max)
Initialize a slide joint.
- `cpConstraint * cpSlideJointNew` (cpBody *a, cpBody *b, cpVect anchorA, cpVect anchorB, cpFloat min, cpFloat max)
Allocate and initialize a slide joint.
- `cpVect cpSlideJointGetAnchorA` (const `cpConstraint` *constraint)
Get the location of the first anchor relative to the first body.
- void `cpSlideJointSetAnchorA` (cpConstraint *constraint, cpVect anchorA)
Set the location of the first anchor relative to the first body.
- `cpVect cpSlideJointGetAnchorB` (const `cpConstraint` *constraint)
Get the location of the second anchor relative to the second body.
- void `cpSlideJointSetAnchorB` (cpConstraint *constraint, cpVect anchorB)
Set the location of the second anchor relative to the second body.
- `cpFloat cpSlideJointGetMin` (const `cpConstraint` *constraint)
Get the minimum distance the joint will maintain between the two anchors.
- void `cpSlideJointSetMin` (cpConstraint *constraint, cpFloat min)
Set the minimum distance the joint will maintain between the two anchors.
- `cpFloat cpSlideJointGetMax` (const `cpConstraint` *constraint)
Get the maximum distance the joint will maintain between the two anchors.
- void `cpSlideJointSetMax` (cpConstraint *constraint, cpFloat max)
Set the maximum distance the joint will maintain between the two anchors.

3.21.1 Detailed Description

Slide joints hold the distance between points on two bodies between a minimum and a maximum.

3.22 cpSpace

Containers for simulating objects in Chipmunk.

Classes

- struct [cpCollisionHandler](#)
Struct that holds function callback pointers to configure custom collision handling.
- struct [cpSpaceDebugColor](#)
Color type to use with the space debug drawing API.
- struct [cpSpaceDebugDrawOptions](#)
Struct used with [cpSpaceDebugDraw\(\)](#) containing drawing callbacks and other drawing settings.

Typedefs

- typedef [cpBool](#)(* [cpCollisionBeginFunc](#)) ([cpArbiter](#) *arb, [cpSpace](#) *space, [cpDataPointer](#) userData)
Collision begin event function callback type.
- typedef [cpBool](#)(* [cpCollisionPreSolveFunc](#)) ([cpArbiter](#) *arb, [cpSpace](#) *space, [cpDataPointer](#) userData)
Collision pre-solve event function callback type.
- typedef void(* [cpCollisionPostSolveFunc](#)) ([cpArbiter](#) *arb, [cpSpace](#) *space, [cpDataPointer](#) userData)
Collision post-solve event function callback type.
- typedef void(* [cpCollisionSeparateFunc](#)) ([cpArbiter](#) *arb, [cpSpace](#) *space, [cpDataPointer](#) userData)
Collision separate event function callback type.
- typedef void(* [cpPostStepFunc](#)) ([cpSpace](#) *space, void *key, void *data)
Post Step callback function type.
- typedef void(* [cpSpacePointQueryFunc](#)) ([cpShape](#) *shape, [cpVect](#) point, [cpFloat](#) distance, [cpVect](#) gradient, void *data)
Nearest point query callback function type.
- typedef void(* [cpSpaceSegmentQueryFunc](#)) ([cpShape](#) *shape, [cpVect](#) point, [cpVect](#) normal, [cpFloat](#) alpha, void *data)
Segment query callback function type.
- typedef void(* [cpSpaceBBQueryFunc](#)) ([cpShape](#) *shape, void *data)
Rectangle Query callback function type.
- typedef void(* [cpSpaceShapeQueryFunc](#)) ([cpShape](#) *shape, [cpContactPointSet](#) *points, void *data)
Shape query callback function type.
- typedef void(* [cpSpaceBodyIteratorFunc](#)) ([cpBody](#) *body, void *data)
Space/body iterator callback function type.
- typedef void(* [cpSpaceShapeIteratorFunc](#)) ([cpShape](#) *shape, void *data)
Space/shape iterator callback function type.
- typedef void(* [cpSpaceConstraintIteratorFunc](#)) ([cpConstraint](#) *constraint, void *data)
Space/constraint iterator callback function type.
- typedef struct [cpSpaceDebugColor](#) [cpSpaceDebugColor](#)
Color type to use with the space debug drawing API.
- typedef void(* [cpSpaceDebugDrawCircleImpl](#)) ([cpVect](#) pos, [cpFloat](#) angle, [cpFloat](#) radius, [cpSpaceDebugColor](#) outlineColor, [cpSpaceDebugColor](#) fillColor, [cpDataPointer](#) data)
Callback type for a function that draws a filled, stroked circle.
- typedef void(* [cpSpaceDebugDrawSegmentImpl](#)) ([cpVect](#) a, [cpVect](#) b, [cpSpaceDebugColor](#) color, [cpDataPointer](#) data)
Callback type for a function that draws a line segment.
- typedef void(* [cpSpaceDebugDrawFatSegmentImpl](#)) ([cpVect](#) a, [cpVect](#) b, [cpFloat](#) radius, [cpSpaceDebugColor](#) outlineColor, [cpSpaceDebugColor](#) fillColor, [cpDataPointer](#) data)

Callback type for a function that draws a thick line segment.

- typedef void(* [cpSpaceDebugDrawPolygonImpl](#)) (int count, const [cpVect](#) *verts, [cpFloat](#) radius, [cpSpaceDebugColor](#) outlineColor, [cpSpaceDebugColor](#) fillColor, [cpDataPointer](#) data)

Callback type for a function that draws a convex polygon.

- typedef void(* [cpSpaceDebugDrawDotImpl](#)) ([cpFloat](#) size, [cpVect](#) pos, [cpSpaceDebugColor](#) color, [cpDataPointer](#) data)

Callback type for a function that draws a dot.

- typedef [cpSpaceDebugColor](#)(* [cpSpaceDebugDrawColorForShapeImpl](#)) ([cpShape](#) *shape, [cpDataPointer](#) data)

Callback type for a function that returns a color for a given shape. This gives you an opportunity to color shapes based on how they are used in your engine.

- typedef enum [cpSpaceDebugDrawFlags](#) **cpSpaceDebugDrawFlags**
- typedef struct [cpSpaceDebugDrawOptions](#) [cpSpaceDebugDrawOptions](#)

Struct used with [cpSpaceDebugDraw\(\)](#) containing drawing callbacks and other drawing settings.

Enumerations

- enum **cpSpaceDebugDrawFlags** { **CP_SPACE_DEBUG_DRAW_SHAPES** = 1<<0, **CP_SPACE_DEBUG_DRAW_CONSTRAINTS** = 1<<1, **CP_SPACE_DEBUG_DRAW_COLLISION_POINTS** = 1<<2 }

Functions

- [cpSpace](#) * [cpSpaceAlloc](#) (void)
Allocate a [cpSpace](#).
- [cpSpace](#) * [cpSpaceInit](#) ([cpSpace](#) *space)
Initialize a [cpSpace](#).
- [cpSpace](#) * [cpSpaceNew](#) (void)
Allocate and initialize a [cpSpace](#).
- void [cpSpaceDestroy](#) ([cpSpace](#) *space)
Destroy a [cpSpace](#).
- void [cpSpaceFree](#) ([cpSpace](#) *space)
Destroy and free a [cpSpace](#).
- int [cpSpaceGetIterations](#) (const [cpSpace](#) *space)
Get number of iterations to use in the impulse solver to solve contacts and other constraints.
- void [cpSpaceSetIterations](#) ([cpSpace](#) *space, int iterations)
Set number of iterations to use in the impulse solver to solve contacts and other constraints.
- [cpVect](#) [cpSpaceGetGravity](#) (const [cpSpace](#) *space)
Get gravity to pass to rigid bodies when integrating velocity.
- void [cpSpaceSetGravity](#) ([cpSpace](#) *space, [cpVect](#) gravity)
Set gravity to pass to rigid bodies when integrating velocity.
- [cpFloat](#) [cpSpaceGetDamping](#) (const [cpSpace](#) *space)
Get the damping rate expressed as the fraction of velocity bodies retain each second.
- void [cpSpaceSetDamping](#) ([cpSpace](#) *space, [cpFloat](#) damping)
Set the damping rate expressed as the fraction of velocity bodies retain each second.
- [cpFloat](#) [cpSpaceGetIdleSpeedThreshold](#) (const [cpSpace](#) *space)
Get speed threshold for a body to be considered idle.
- void [cpSpaceSetIdleSpeedThreshold](#) ([cpSpace](#) *space, [cpFloat](#) idleSpeedThreshold)
Set speed threshold for a body to be considered idle.
- [cpFloat](#) [cpSpaceGetSleepTimeThreshold](#) (const [cpSpace](#) *space)
Get the time a group of bodies must remain idle in order to fall asleep.
- void [cpSpaceSetSleepTimeThreshold](#) ([cpSpace](#) *space, [cpFloat](#) sleepTimeThreshold)

- Set the time a group of bodies must remain idle in order to fall asleep.*

 - `cpFloat cpSpaceGetCollisionSlop` (`const cpSpace *space`)
- Get amount of encouraged penetration between colliding shapes.*

 - `void cpSpaceSetCollisionSlop` (`cpSpace *space`, `cpFloat collisionSlop`)
- Set amount of encouraged penetration between colliding shapes.*

 - `cpFloat cpSpaceGetCollisionBias` (`const cpSpace *space`)
- Get how fast overlapping shapes are pushed apart.*

 - `void cpSpaceSetCollisionBias` (`cpSpace *space`, `cpFloat collisionBias`)
- Set how fast overlapping shapes are pushed apart.*

 - `cpTimestamp cpSpaceGetCollisionPersistence` (`const cpSpace *space`)
- Get number of frames that contact information should persist.*

 - `void cpSpaceSetCollisionPersistence` (`cpSpace *space`, `cpTimestamp collisionPersistence`)
- Set number of frames that contact information should persist.*

 - `cpDataPointer cpSpaceGetUserData` (`const cpSpace *space`)
- Get user definable data pointer.*

 - `void cpSpaceSetUserData` (`cpSpace *space`, `cpDataPointer userData`)
- Set user definable data pointer.*

 - `cpBody * cpSpaceGetStaticBody` (`const cpSpace *space`)
- The Space provided static body for a given cpSpace.*

 - `cpFloat cpSpaceGetCurrentTimeStep` (`const cpSpace *space`)
- Returns the current (or most recent) time step used with the given space.*

 - `cpBool cpSpaceIsLocked` (`cpSpace *space`)
- returns true from inside a callback when objects cannot be added/removed.*

 - `cpCollisionHandler * cpSpaceAddDefaultCollisionHandler` (`cpSpace *space`)
- Create or return the existing collision handler that is called for all collisions that are not handled by a more specific collision handler.*

 - `cpCollisionHandler * cpSpaceAddCollisionHandler` (`cpSpace *space`, `cpCollisionType a`, `cpCollisionType b`)
- Create or return the existing collision handler for the specified pair of collision types.*

 - `cpCollisionHandler * cpSpaceAddWildcardHandler` (`cpSpace *space`, `cpCollisionType type`)
- Create or return the existing wildcard collision handler for the specified type.*

 - `cpShape * cpSpaceAddShape` (`cpSpace *space`, `cpShape *shape`)
- Add a collision shape to the simulation.*

 - `cpBody * cpSpaceAddBody` (`cpSpace *space`, `cpBody *body`)
- Add a rigid body to the simulation.*

 - `cpConstraint * cpSpaceAddConstraint` (`cpSpace *space`, `cpConstraint *constraint`)
- Add a constraint to the simulation.*

 - `void cpSpaceRemoveShape` (`cpSpace *space`, `cpShape *shape`)
- Remove a collision shape from the simulation.*

 - `void cpSpaceRemoveBody` (`cpSpace *space`, `cpBody *body`)
- Remove a rigid body from the simulation.*

 - `void cpSpaceRemoveConstraint` (`cpSpace *space`, `cpConstraint *constraint`)
- Remove a constraint from the simulation.*

 - `cpBool cpSpaceContainsShape` (`cpSpace *space`, `cpShape *shape`)
- Test if a collision shape has been added to the space.*

 - `cpBool cpSpaceContainsBody` (`cpSpace *space`, `cpBody *body`)
- Test if a rigid body has been added to the space.*

 - `cpBool cpSpaceContainsConstraint` (`cpSpace *space`, `cpConstraint *constraint`)
- Test if a constraint has been added to the space.*

 - `cpBool cpSpaceAddPostStepCallback` (`cpSpace *space`, `cpPostStepFunc func`, `void *key`, `void *data`)
- Schedule a post-step callback to be called when cpSpaceStep() finishes.*

- void `cpSpacePointQuery` (`cpSpace` *space, `cpVect` point, `cpFloat` maxDistance, `cpShapeFilter` filter, `cpSpacePointQueryFunc` func, void *data)
Query the space at a point and call `func` for each shape found.
- `cpShape` * `cpSpacePointQueryNearest` (`cpSpace` *space, `cpVect` point, `cpFloat` maxDistance, `cpShapeFilter` filter, `cpPointQueryInfo` *out)
Query the space at a point and return the nearest shape found. Returns NULL if no shapes were found.
- void `cpSpaceSegmentQuery` (`cpSpace` *space, `cpVect` start, `cpVect` end, `cpFloat` radius, `cpShapeFilter` filter, `cpSpaceSegmentQueryFunc` func, void *data)
Perform a directed line segment query (like a raycast) against the space calling `func` for each shape intersected.
- `cpShape` * `cpSpaceSegmentQueryFirst` (`cpSpace` *space, `cpVect` start, `cpVect` end, `cpFloat` radius, `cpShapeFilter` filter, `cpSegmentQueryInfo` *out)
Perform a directed line segment query (like a raycast) against the space and return the first shape hit. Returns NULL if no shapes were hit.
- void `cpSpaceBBQuery` (`cpSpace` *space, `cpBB` bb, `cpShapeFilter` filter, `cpSpaceBBQueryFunc` func, void *data)
Perform a fast rectangle query on the space calling `func` for each shape found.
- `cpBool` `cpSpaceShapeQuery` (`cpSpace` *space, `cpShape` *shape, `cpSpaceShapeQueryFunc` func, void *data)
Query a space for any shapes overlapping the given shape and call `func` for each shape found.
- void `cpSpaceEachBody` (`cpSpace` *space, `cpSpaceBodyIteratorFunc` func, void *data)
Call `func` for each body in the space.
- void `cpSpaceEachShape` (`cpSpace` *space, `cpSpaceShapeIteratorFunc` func, void *data)
Call `func` for each shape in the space.
- void `cpSpaceEachConstraint` (`cpSpace` *space, `cpSpaceConstraintIteratorFunc` func, void *data)
Call `func` for each constraint in the space.
- void `cpSpaceReindexStatic` (`cpSpace` *space)
Update the collision detection info for the static shapes in the space.
- void `cpSpaceReindexShape` (`cpSpace` *space, `cpShape` *shape)
Update the collision detection data for a specific shape in the space.
- void `cpSpaceReindexShapesForBody` (`cpSpace` *space, `cpBody` *body)
Update the collision detection data for all shapes attached to a body.
- void `cpSpaceUseSpatialHash` (`cpSpace` *space, `cpFloat` dim, int count)
Switch the space to use a spatial hash as it's spatial index.
- void `cpSpaceStep` (`cpSpace` *space, `cpFloat` dt)
Step the space forward in time by `dt`.
- void `cpSpaceDebugDraw` (`cpSpace` *space, `cpSpaceDebugDrawOptions` *options)
Debug draw the current state of the space using the supplied drawing options.

3.22.1 Detailed Description

Containers for simulating objects in Chipmunk.

Controls how all the rigid bodies, shapes and constraints interact together.

3.22.2 Typedef Documentation

3.22.2.1 typedef `cpBool`(* `cpCollisionBeginFunc`) (`cpArbiter` *arb, `cpSpace` *space, `cpDataPointer` userData)

Collision begin event function callback type.

Returning false from a begin callback causes the collision to be ignored until the the separate callback is called when the objects stop colliding.

3.22.2.2 `typedef cpBool(* cpCollisionPreSolveFunc)(cpArbiter *arb, cpSpace *space, cpDataPointer userData)`

Collision pre-solve event function callback type.

Returning false from a pre-step callback causes the collision to be ignored until the next step.

3.22.3 Function Documentation

3.22.3.1 `cpCollisionHandler* cpSpaceAddCollisionHandler (cpSpace * space, cpCollisionType a, cpCollisionType b)`

Create or return the existing collision handler for the specified pair of collision types.

If wildcard handlers are used with either of the collision types, it's the responsibility of the custom handler to invoke the wildcard handlers.

3.22.3.2 `cpBool cpSpaceAddPostStepCallback (cpSpace * space, cpPostStepFunc func, void * key, void * data)`

Schedule a post-step callback to be called when [cpSpaceStep\(\)](#) finishes.

You can only register one callback per unique value for `key`. Returns true only if `key` has never been scheduled before. It's possible to pass `NULL` for `func` if you only want to mark `key` as being used.

3.22.3.3 `cpShape* cpSpaceAddShape (cpSpace * space, cpShape * shape)`

Add a collision shape to the simulation.

If the shape is attached to a static body, it will be added as a static shape.

3.22.3.4 `void cpSpaceBBQuery (cpSpace * space, cpBB bb, cpShapeFilter filter, cpSpaceBBQueryFunc func, void * data)`

Perform a fast rectangle query on the space calling `func` for each shape found.

Only the shape's bounding boxes are checked for overlap, not their full shape.

3.22.3.5 `cpFloat cpSpaceGetCollisionBias (const cpSpace * space)`

Get how fast overlapping shapes are pushed apart.

Expressed as a fraction of the error remaining after each second. Defaults to `pow(1.0 - 0.1, 60.0)` meaning that Chipmunk fixes 10% of overlap each frame at 60Hz.

3.22.3.6 `cpTimestamp cpSpaceGetCollisionPersistence (const cpSpace * space)`

Get number of frames that contact information should persist.

Defaults to 3. There is probably never a reason to change this value.

3.22.3.7 `cpFloat cpSpaceGetCollisionSlop (const cpSpace * space)`

Get amount of encouraged penetration between colliding shapes.

Used to reduce oscillating contacts and keep the collision cache warm. Defaults to 0.1. If you have poor simulation quality, increase this number as much as possible without allowing visible amounts of overlap.

3.22.3.8 `cpFloat cpSpaceGetCurrentTimeStep (const cpSpace * space)`

Returns the current (or most recent) time step used with the given space.

Useful from callbacks if your time step is not a compile-time global.

3.22.3.9 `cpFloat cpSpaceGetDamping (const cpSpace * space)`

Get the damping rate expressed as the fraction of velocity bodies retain each second.

A value of 0.9 would mean that each body's velocity will drop 10% per second. The default value is 1.0, meaning no damping is applied.

Note

This damping value is different than those of [cpDampedSpring](#) and [cpDampedRotarySpring](#).

3.22.3.10 `cpFloat cpSpaceGetIdleSpeedThreshold (const cpSpace * space)`

Get speed threshold for a body to be considered idle.

The default value of 0 means to let the space guess a good threshold based on gravity.

3.22.3.11 `cpFloat cpSpaceGetSleepTimeThreshold (const cpSpace * space)`

Get the time a group of bodies must remain idle in order to fall asleep.

Enabling sleeping also implicitly enables the the contact graph. The default value of INFINITY disables the sleeping algorithm.

3.22.3.12 `cpBody* cpSpaceGetStaticBody (const cpSpace * space)`

The Space provided static body for a given [cpSpace](#).

This is merely provided for convenience and you are not required to use it.

3.22.3.13 `cpDataPointer cpSpaceGetUserData (const cpSpace * space)`

Get user definable data pointer.

Generally this points to your game's controller or game state class so you can access it when given a [cpSpace](#) reference in a callback.

3.22.3.14 `void cpSpaceSetCollisionBias (cpSpace * space, cpFloat collisionBias)`

Set how fast overlapping shapes are pushed apart.

Expressed as a fraction of the error remaining after each second. Defaults to $\text{pow}(1.0 - 0.1, 60.0)$ meaning that Chipmunk fixes 10% of overlap each frame at 60Hz.

3.22.3.15 `void cpSpaceSetCollisionPersistence (cpSpace * space, cpTimestamp collisionPersistence)`

Set number of frames that contact information should persist.

Defaults to 3. There is probably never a reason to change this value.

3.22.3.16 void cpSpaceSetCollisionSlop (cpSpace * *space*, cpFloat *collisionSlop*)

Set amount of encouraged penetration between colliding shapes.

Used to reduce oscillating contacts and keep the collision cache warm. Defaults to 0.1. If you have poor simulation quality, increase this number as much as possible without allowing visible amounts of overlap.

3.22.3.17 void cpSpaceSetDamping (cpSpace * *space*, cpFloat *damping*)

Set the damping rate expressed as the fraction of velocity bodies retain each second.

A value of 0.9 would mean that each body's velocity will drop 10% per second. The default value is 1.0, meaning no damping is applied.

Note

This damping value is different than those of [cpDampedSpring](#) and [cpDampedRotarySpring](#).

3.22.3.18 void cpSpaceSetIdleSpeedThreshold (cpSpace * *space*, cpFloat *idleSpeedThreshold*)

Set speed threshold for a body to be considered idle.

The default value of 0 means to let the space guess a good threshold based on gravity.

3.22.3.19 void cpSpaceSetSleepTimeThreshold (cpSpace * *space*, cpFloat *sleepTimeThreshold*)

Set the time a group of bodies must remain idle in order to fall asleep.

Enabling sleeping also implicitly enables the the contact graph. The default value of INFINITY disables the sleeping algorithm.

3.22.3.20 void cpSpaceSetUserData (cpSpace * *space*, cpDataPointer *userData*)

Set user definable data pointer.

Generally this points to your game's controller or game state class so you can access it when given a [cpSpace](#) reference in a callback.

3.23 cpSpatialIndex

Spatial indexes are data structures that are used to accelerate collision detection and spatial queries.

Classes

- struct [cpSpatialIndexClass](#)
Used to accelerate collision detection.

Typedefs

- typedef [cpBB](#)(* [cpSpatialIndexBBFunc](#)) (void *obj)
Spatial index bounding box callback function type.
- typedef void(* [cpSpatialIndexIteratorFunc](#)) (void *obj, void *data)
Spatial index/object iterator callback function type.
- typedef [cpCollisionID](#)(* [cpSpatialIndexQueryFunc](#)) (void *obj1, void *obj2, [cpCollisionID](#) id, void *data)
Spatial query callback function type.
- typedef [cpFloat](#)(* [cpSpatialIndexSegmentQueryFunc](#)) (void *obj1, void *obj2, void *data)
Spatial segment query callback function type.
- typedef struct [cpSpatialIndexClass](#) **cpSpatialIndexClass**
- typedef struct cpSpatialIndex **cpSpatialIndex**
- typedef struct cpSpaceHash **cpSpaceHash**
- typedef struct cpBBTree **cpBBTree**
- typedef [cpVect](#)(* [cpBBTreeVelocityFunc](#)) (void *obj)
Bounding box tree velocity callback function.
- typedef struct cpSweep1D **cpSweep1D**
- typedef void(* [cpSpatialIndexDestroyImpl](#)) (cpSpatialIndex *index)
- typedef int(* [cpSpatialIndexCountImpl](#)) (cpSpatialIndex *index)
- typedef void(* [cpSpatialIndexEachImpl](#)) (cpSpatialIndex *index, [cpSpatialIndexIteratorFunc](#) func, void *data)
- typedef [cpBool](#)(* [cpSpatialIndexContainsImpl](#)) (cpSpatialIndex *index, void *obj, [cpHashValue](#) hashid)
- typedef void(* [cpSpatialIndexInsertImpl](#)) (cpSpatialIndex *index, void *obj, [cpHashValue](#) hashid)
- typedef void(* [cpSpatialIndexRemoveImpl](#)) (cpSpatialIndex *index, void *obj, [cpHashValue](#) hashid)
- typedef void(* [cpSpatialIndexReindexImpl](#)) (cpSpatialIndex *index)
- typedef void(* [cpSpatialIndexReindexObjectImpl](#)) (cpSpatialIndex *index, void *obj, [cpHashValue](#) hashid)
- typedef void(* [cpSpatialIndexReindexQueryImpl](#)) (cpSpatialIndex *index, [cpSpatialIndexQueryFunc](#) func, void *data)
- typedef void(* [cpSpatialIndexQueryImpl](#)) (cpSpatialIndex *index, void *obj, [cpBB](#) bb, [cpSpatialIndexQueryFunc](#) func, void *data)
- typedef void(* [cpSpatialIndexSegmentQueryImpl](#)) (cpSpatialIndex *index, void *obj, [cpVect](#) a, [cpVect](#) b, [cpFloat](#) t_exit, [cpSpatialIndexSegmentQueryFunc](#) func, void *data)

Functions

- cpSpaceHash * [cpSpaceHashAlloc](#) (void)
Allocate a spatial hash.
- cpSpatialIndex * [cpSpaceHashInit](#) (cpSpaceHash *hash, [cpFloat](#) celldim, int numcells, [cpSpatialIndexBBFunc](#) bbfunc, cpSpatialIndex *staticIndex)
Initialize a spatial hash.
- cpSpatialIndex * [cpSpaceHashNew](#) ([cpFloat](#) celldim, int cells, [cpSpatialIndexBBFunc](#) bbfunc, cpSpatialIndex *staticIndex)
Allocate and initialize a spatial hash.

- void `cpSpaceHashResize` (cpSpaceHash *hash, cpFloat celldim, int numcells)
Change the cell dimensions and table size of the spatial hash to tune it.
- cpBBTree * `cpBBTreeAlloc` (void)
Allocate a bounding box tree.
- cpSpatialIndex * `cpBBTreeInit` (cpBBTree *tree, cpSpatialIndexBBFunc bbfunc, cpSpatialIndex *staticIndex)
Initialize a bounding box tree.
- cpSpatialIndex * `cpBBTreeNew` (cpSpatialIndexBBFunc bbfunc, cpSpatialIndex *staticIndex)
Allocate and initialize a bounding box tree.
- void `cpBBTreeOptimize` (cpSpatialIndex *index)
Perform a static top down optimization of the tree.
- void `cpBBTreeSetVelocityFunc` (cpSpatialIndex *index, cpBBTreeVelocityFunc func)
Set the velocity function for the bounding box tree to enable temporal coherence.
- cpSweep1D * `cpSweep1DAlloc` (void)
Allocate a 1D sort and sweep broadphase.
- cpSpatialIndex * `cpSweep1DInit` (cpSweep1D *sweep, cpSpatialIndexBBFunc bbfunc, cpSpatialIndex *staticIndex)
Initialize a 1D sort and sweep broadphase.
- cpSpatialIndex * `cpSweep1DNew` (cpSpatialIndexBBFunc bbfunc, cpSpatialIndex *staticIndex)
Allocate and initialize a 1D sort and sweep broadphase.
- void `cpSpatialIndexFree` (cpSpatialIndex *index)
Destroy and free a spatial index.
- void `cpSpatialIndexCollideStatic` (cpSpatialIndex *dynamicIndex, cpSpatialIndex *staticIndex, cpSpatialIndexQueryFunc func, void *data)
Collide the objects in dynamicIndex against the objects in staticIndex using the query callback function.
- static void `cpSpatialIndexDestroy` (cpSpatialIndex *index)
Destroy a spatial index.
- static int `cpSpatialIndexCount` (cpSpatialIndex *index)
Get the number of objects in the spatial index.
- static void `cpSpatialIndexEach` (cpSpatialIndex *index, cpSpatialIndexIteratorFunc func, void *data)
Iterate the objects in the spatial index. func will be called once for each object.
- static cpBool `cpSpatialIndexContains` (cpSpatialIndex *index, void *obj, cpHashValue hashid)
Returns true if the spatial index contains the given object.
- static void `cpSpatialIndexInsert` (cpSpatialIndex *index, void *obj, cpHashValue hashid)
Add an object to a spatial index.
- static void `cpSpatialIndexRemove` (cpSpatialIndex *index, void *obj, cpHashValue hashid)
Remove an object from a spatial index.
- static void `cpSpatialIndexReindex` (cpSpatialIndex *index)
Perform a full reindex of a spatial index.
- static void `cpSpatialIndexReindexObject` (cpSpatialIndex *index, void *obj, cpHashValue hashid)
Reindex a single object in the spatial index.
- static void `cpSpatialIndexQuery` (cpSpatialIndex *index, void *obj, cpBB bb, cpSpatialIndexQueryFunc func, void *data)
Perform a rectangle query against the spatial index, calling func for each potential match.
- static void `cpSpatialIndexSegmentQuery` (cpSpatialIndex *index, void *obj, cpVect a, cpVect b, cpFloat t_exit, cpSpatialIndexSegmentQueryFunc func, void *data)
Perform a segment query against the spatial index, calling func for each potential match.
- static void `cpSpatialIndexReindexQuery` (cpSpatialIndex *index, cpSpatialIndexQueryFunc func, void *data)
Simultaneously reindex and find all colliding objects.

3.23.1 Detailed Description

Spatial indexes are data structures that are used to accelerate collision detection and spatial queries.

Chipmunk provides a number of spatial index algorithms to pick from and they are programmed in a generic way so that you can use them for holding more than just [cpShape](#) structs.

It works by using `void` pointers to the objects you add and using a callback to ask your code for bounding boxes when it needs them. Several types of queries can be performed on an index as well as reindexing and full collision information. All communication to the spatial indexes is performed through callback functions.

Spatial indexes should be treated as opaque structs. This means you shouldn't be reading any of the struct fields.

3.23.2 Typedef Documentation

3.23.2.1 `typedef cpVect(* cpBBTreeVelocityFunc) (void *obj)`

Bounding box tree velocity callback function.

This function should return an estimate for the object's velocity.

3.23.2.2 `typedef cpBB(* cpSpatialIndexBBFunc) (void *obj)`

Spatial index bounding box callback function type.

The spatial index calls this function and passes you a pointer to an object you added when it needs to get the bounding box associated with that object.

3.23.3 Function Documentation

3.23.3.1 `void cpSpaceHashResize (cpSpaceHash * hash, cpFloat celldim, int numcells)`

Change the cell dimensions and table size of the spatial hash to tune it.

The cell dimensions should roughly match the average size of your objects and the table size should be ~10 larger than the number of objects inserted. Some trial and error is required to find the optimum numbers for efficiency.

3.23.3.2 `static cpBool cpSpatialIndexContains (cpSpatialIndex * index, void * obj, cpHashValue hashid) [inline], [static]`

Returns true if the spatial index contains the given object.

Most spatial indexes use hashed storage, so you must provide a hash value too.

3.23.3.3 `static void cpSpatialIndexInsert (cpSpatialIndex * index, void * obj, cpHashValue hashid) [inline], [static]`

Add an object to a spatial index.

Most spatial indexes use hashed storage, so you must provide a hash value too.

3.23.3.4 `static void cpSpatialIndexReindexQuery (cpSpatialIndex * index, cpSpatialIndexQueryFunc func, void * data) [inline], [static]`

Simultaneously reindex and find all colliding objects.

`func` will be called once for each potentially overlapping pair of objects found. If the spatial index was initialized with a static index, it will collide its objects against that as well.


```
3.23.3.5 static void cpSpatialIndexRemove ( cpSpatialIndex * index, void * obj, cpHashValue hashid ) [inline],  
[static]
```

Remove an object from a spatial index.

Most spatial indexes use hashed storage, so you must provide a hash value too.

3.24 cpVect

Chipmunk's 2D vector type along with a handy 2D vector math lib.

Functions

- static `cpVect cpv` (const `cpFloat` x, const `cpFloat` y)
Convenience constructor for `cpVect` structs.
- static `cpBool cpveql` (const `cpVect` v1, const `cpVect` v2)
Check if two vectors are equal. (Be careful when comparing floating point numbers!)
- static `cpVect cpvadd` (const `cpVect` v1, const `cpVect` v2)
Add two vectors.
- static `cpVect cpvsub` (const `cpVect` v1, const `cpVect` v2)
Subtract two vectors.
- static `cpVect cpvneg` (const `cpVect` v)
Negate a vector.
- static `cpVect cpvmult` (const `cpVect` v, const `cpFloat` s)
Scalar multiplication.
- static `cpFloat cpvdot` (const `cpVect` v1, const `cpVect` v2)
Vector dot product.
- static `cpFloat cpvcross` (const `cpVect` v1, const `cpVect` v2)
2D vector cross product analog.
- static `cpVect cpvperp` (const `cpVect` v)
Returns a perpendicular vector. (90 degree rotation)
- static `cpVect cpvrperp` (const `cpVect` v)
Returns a perpendicular vector. (-90 degree rotation)
- static `cpVect cpvproject` (const `cpVect` v1, const `cpVect` v2)
Returns the vector projection of v1 onto v2.
- static `cpVect cpvforangle` (const `cpFloat` a)
Returns the unit length vector for the given angle (in radians).
- static `cpFloat cpvtoangle` (const `cpVect` v)
Returns the angular direction v is pointing in (in radians).
- static `cpVect cpvrotate` (const `cpVect` v1, const `cpVect` v2)
Uses complex number multiplication to rotate v1 by v2. Scaling will occur if v1 is not a unit vector.
- static `cpVect cpvunrotate` (const `cpVect` v1, const `cpVect` v2)
Inverse of `cpvrotate()`.
- static `cpFloat cpvlengthsq` (const `cpVect` v)
Returns the squared length of v. Faster than `cpvlength()` when you only need to compare lengths.
- static `cpFloat cpvlength` (const `cpVect` v)
Returns the length of v.
- static `cpVect cpvlerp` (const `cpVect` v1, const `cpVect` v2, const `cpFloat` t)
Linearly interpolate between v1 and v2.
- static `cpVect cpvnormalize` (const `cpVect` v)
Returns a normalized copy of v.
- static `cpVect cpvslerp` (const `cpVect` v1, const `cpVect` v2, const `cpFloat` t)
Spherical linearly interpolate between v1 and v2.
- static `cpVect cpvslerpconst` (const `cpVect` v1, const `cpVect` v2, const `cpFloat` a)
Spherical linearly interpolate between v1 towards v2 by no more than angle a radians.
- static `cpVect cpvclamp` (const `cpVect` v, const `cpFloat` len)
Clamp v to length len.

- static `cpVect cpvlerpconst (cpVect v1, cpVect v2, cpFloat d)`
Linearly interpolate between v1 towards v2 by distance d.
- static `cpFloat cpvdist (const cpVect v1, const cpVect v2)`
Returns the distance between v1 and v2.
- static `cpFloat cpvdistsq (const cpVect v1, const cpVect v2)`
Returns the squared distance between v1 and v2. Faster than `cpvdist()` when you only need to compare distances.
- static `cpBool cpvnear (const cpVect v1, const cpVect v2, const cpFloat dist)`
Returns true if the distance between v1 and v2 is less than dist.

Variables

- static const `cpVect cpvzero = {0.0f,0.0f}`
Constant for the zero vector.

3.24.1 Detailed Description

Chipmunk's 2D vector type along with a handy 2D vector math lib.

Chipmunk's 2D vector type.

3.24.2 Function Documentation

3.24.2.1 static cpFloat cpvcross (const cpVect v1, const cpVect v2) [inline],[static]

2D vector cross product analog.

The cross product of 2D vectors results in a 3D vector with only a z component. This function returns the magnitude of the z value.

3.25 cpMat2x2

2x2 matrix type used for tensors and such.

Functions

- static [cpMat2x2 cpMat2x2New](#) ([cpFloat](#) a, [cpFloat](#) b, [cpFloat](#) c, [cpFloat](#) d)
Convenience constructor for [cpMat2x2](#) structs.
- static [cpVect cpMat2x2Transform](#) ([cpMat2x2](#) m, [cpVect](#) v)
Multiply the matrix with a vector.

3.25.1 Detailed Description

2x2 matrix type used for tensors and such.

Chapter 4

Class Documentation

4.1 cpArbiter Struct Reference

Tracks pairs of colliding shapes.

```
#include <cpArbiter_private.h>
```

Public Attributes

- cpFloat **e**
- cpFloat **u**
- cpVect **surface_vr**
- cpDataPointer **data**
- const cpShape * **a**
- const cpShape * **b**
- cpBody * **body_a**
- cpBody * **body_b**
- struct cpArbiterThread thread_a **thread_b**
- int **count**
- struct cpContact * **contacts**
- cpVect **n**
- cpCollisionHandler * **handler**
- cpCollisionHandler * **handlerA**
- cpCollisionHandler * **handlerB**
- cpBool **swapped**
- cpTimestamp **stamp**
- enum cpArbiterState **state**

4.1.1 Detailed Description

Tracks pairs of colliding shapes.

The documentation for this struct was generated from the following file:

- cpArbiter_private.h

4.2 cpArbiterThread Struct Reference

A doubly linked list for the cpArbiter values.

```
#include <cpArbiter_private.h>
```

Public Attributes

- struct [cpArbiter](#) * **next**
- struct [cpArbiter](#) * **prev**

4.2.1 Detailed Description

A doubly linked list for the [cpArbiter](#) values.

The documentation for this struct was generated from the following file:

- cpArbiter_private.h

4.3 cpArray Struct Reference

Chipmunk's array data structure.

```
#include <cpArray_private.h>
```

Public Attributes

- int **num**
- int **max**
- void ** **arr**

4.3.1 Detailed Description

Chipmunk's array data structure.

The documentation for this struct was generated from the following file:

- cpArray_private.h

4.4 cpBB Struct Reference

Chipmunk's axis-aligned 2D bounding box type. (left, bottom, right, top)

```
#include <cpBB.h>
```

Public Attributes

- [cpFloat](#) **l**
- [cpFloat](#) **b**
- [cpFloat](#) **r**
- [cpFloat](#) **t**

4.4.1 Detailed Description

Chipmunk's axis-aligned 2D bounding box type. (left, bottom, right, top)

The documentation for this struct was generated from the following file:

- cpBB.h

4.5 cpBody Struct Reference

Chipmunk's rigid body type.

```
#include <cpBody_private.h>
```

Public Attributes

- [cpBodyVelocityFunc](#) **velocity_func**
- [cpBodyPositionFunc](#) **position_func**
- [cpFloat](#) **m**
- [cpFloat](#) **m_inv**
- [cpFloat](#) **i**
- [cpFloat](#) **i_inv**
- [cpVect](#) **cog**
- [cpVect](#) **p**
- [cpVect](#) **v**
- [cpVect](#) **f**
- [cpFloat](#) **a**
- [cpFloat](#) **w**
- [cpFloat](#) **t**
- [cpTransform](#) **transform**
- [cpDataPointer](#) **userData**
- [cpVect](#) **v_bias**
- [cpFloat](#) **w_bias**
- [cpSpace](#) * **space**
- [cpShape](#) * **shapeList**
- [cpArbiter](#) * **arbiterList**
- [cpConstraint](#) * **constraintList**
- struct {
 - [cpBody](#) * **root**
 - [cpBody](#) * **next**
 - [cpFloat](#) **idleTime**
- } **sleeping**

4.5.1 Detailed Description

Chipmunk's rigid body type.

The documentation for this struct was generated from the following file:

- [cpBody_private.h](#)

4.6 cpCircleShape Struct Reference

A perfect circle shape.

```
#include <cpShape_private.h>
```

Public Attributes

- [cpShape](#) **shape**
- [cpVect](#) **c**
- [cpVect](#) **tc**
- [cpFloat](#) **r**

4.6.1 Detailed Description

A perfect circle shape.

The documentation for this struct was generated from the following file:

- cpShape_private.h

4.7 cpCollisionHandler Struct Reference

Struct that holds function callback pointers to configure custom collision handling.

```
#include <cpSpace.h>
```

Public Attributes

- const [cpCollisionType](#) typeA
Collision type identifier of the first shape that this handler recognizes.
- const [cpCollisionType](#) typeB
Collision type identifier of the second shape that this handler recognizes.
- [cpCollisionBeginFunc](#) beginFunc
This function is called when two shapes with types that match this collision handler begin colliding.
- [cpCollisionPreSolveFunc](#) preSolveFunc
This function is called each step when two shapes with types that match this collision handler are colliding.
- [cpCollisionPostSolveFunc](#) postSolveFunc
This function is called each step when two shapes with types that match this collision handler are colliding.
- [cpCollisionSeparateFunc](#) separateFunc
This function is called when two shapes with types that match this collision handler stop colliding.
- [cpDataPointer](#) userData
This is a user definable context pointer that is passed to all of the collision handler functions.

4.7.1 Detailed Description

Struct that holds function callback pointers to configure custom collision handling.

Collision handlers have a pair of types; when a collision occurs between two shapes that have these types, the collision handler functions are triggered.

4.7.2 Member Data Documentation

4.7.2.1 cpCollisionPostSolveFunc cpCollisionHandler::postSolveFunc

This function is called each step when two shapes with types that match this collision handler are colliding.

It's called after the collision solver runs so that you can read back information about the collision to trigger events in your game.

4.7.2.2 cpCollisionPreSolveFunc cpCollisionHandler::preSolveFunc

This function is called each step when two shapes with types that match this collision handler are colliding.

It's called before the collision solver runs so that you can affect a collision's outcome.

4.7.2.3 const cpCollisionType cpCollisionHandler::typeA

Collision type identifier of the first shape that this handler recognizes.

In the collision handler callback, the shape with this type will be the first argument. Read only.

4.7.2.4 const cpCollisionType cpCollisionHandler::typeB

Collision type identifier of the second shape that this handler recognizes.

In the collision handler callback, the shape with this type will be the second argument. Read only.

The documentation for this struct was generated from the following file:

- cpSpace.h

4.8 cpCollisionInfo Struct Reference

Holds information about the collision.

```
#include <cpArbiter_private.h>
```

Public Attributes

- const cpShape * **a**
- const cpShape * **b**
- cpCollisionID **id**
- cpVect **n**
- int **count**
- struct cpContact * **arr**

4.8.1 Detailed Description

Holds information about the collision.

The documentation for this struct was generated from the following file:

- cpArbiter_private.h

4.9 cpConstraint Struct Reference

Constraints connect two cpBody objects together.

```
#include <cpConstraint_private.h>
```

Public Attributes

- const cpConstraintClass * **klass**
- cpSpace * **space**
- cpBody * **a**
- cpBody * **b**
- cpConstraint * **next_a**
- cpConstraint * **next_b**
- cpFloat **maxForce**

- [cpFloat](#) **errorBias**
- [cpFloat](#) **maxBias**
- [cpBool](#) **collideBodies**
- [cpConstraintPreSolveFunc](#) **preSolve**
- [cpConstraintPostSolveFunc](#) **postSolve**
- [cpDataPointer](#) **userData**

4.9.1 Detailed Description

Constraints connect two [cpBody](#) objects together.

[cpConstraint](#) is the base constraint struct that the other constraints build off of.

The documentation for this struct was generated from the following file:

- [cpConstraint_private.h](#)

4.10 cpConstraintClass Struct Reference

Struct that holds function callback pointers for constraints.

```
#include <cpConstraint_private.h>
```

Public Attributes

- [cpConstraintPreStepImpl](#) **preStep**
- [cpConstraintApplyCachedImpulseImpl](#) **applyCachedImpulse**
- [cpConstraintApplyImpulseImpl](#) **applyImpulse**
- [cpConstraintGetImpulseImpl](#) **getImpulse**

4.10.1 Detailed Description

Struct that holds function callback pointers for constraints.

The documentation for this struct was generated from the following file:

- [cpConstraint_private.h](#)

4.11 cpContact Struct Reference

Holds information about the contact points of the collision.

```
#include <cpArbiter_private.h>
```

Public Attributes

- [cpVect](#) **r1**
- [cpVect](#) **r2**
- [cpFloat](#) **nMass**
- [cpFloat](#) **tMass**
- [cpFloat](#) **bounce**
- [cpFloat](#) **jnAcc**
- [cpFloat](#) **jtAcc**

- [cpFloat](#) **jBias**
- [cpFloat](#) **bias**
- [cpHashValue](#) **hash**

4.11.1 Detailed Description

Holds information about the contact points of the collision.

The documentation for this struct was generated from the following file:

- cpArbiter_private.h

4.12 cpContactPointSet Struct Reference

A struct that wraps up the important collision data for an arbiter.

```
#include <cpArbiter.h>
```

Public Attributes

- int [count](#)
The number of contact points in the set.
- [cpVect](#) **normal**
The normal of the collision.
- struct {
 [cpVect](#) **pointA**
 The position of the contact on the surface of each shape.
 [cpVect](#) **pointB**
 [cpFloat](#) **distance**
 Penetration distance of the two shapes.
} [points](#) [CP_MAX_CONTACTS_PER_ARBITER]

The array of contact points.

4.12.1 Detailed Description

A struct that wraps up the important collision data for an arbiter.

4.12.2 Member Data Documentation

4.12.2.1 [cpFloat](#) cpContactPointSet::distance

Penetration distance of the two shapes.

Overlapping means it will be negative. This value is calculated as `cpvdot(cpvsub(point2, point1), normal)` and is ignored by `cpArbiterSetContactPointSet()`.

The documentation for this struct was generated from the following file:

- cpArbiter.h

4.13 cpDampedRotarySpring Struct Reference

Like a [cpDampedSpring](#), but operates in a rotational fashion.

```
#include <cpConstraint_private.h>
```

Public Attributes

- [cpConstraint](#) **constraint**
- [cpFloat](#) **restAngle**
- [cpFloat](#) **stiffness**
- [cpFloat](#) **damping**
- [cpDampedRotarySpringTorqueFunc](#) **springTorqueFunc**
- [cpFloat](#) **target_wrn**
- [cpFloat](#) **w_coef**
- [cpFloat](#) **iSum**
- [cpFloat](#) **jAcc**

4.13.1 Detailed Description

Like a [cpDampedSpring](#), but operates in a rotational fashion.

The documentation for this struct was generated from the following file:

- [cpConstraint_private.h](#)

4.14 cpDampedSpring Struct Reference

A spring with a damper.

```
#include <cpConstraint_private.h>
```

Public Attributes

- [cpConstraint](#) **constraint**
- [cpVect](#) **anchorA**
- [cpVect](#) **anchorB**
- [cpFloat](#) **restLength**
- [cpFloat](#) **stiffness**
- [cpFloat](#) **damping**
- [cpDampedSpringForceFunc](#) **springForceFunc**
- [cpFloat](#) **target_vrn**
- [cpFloat](#) **v_coef**
- [cpVect](#) **r1**
- [cpVect](#) **r2**
- [cpFloat](#) **nMass**
- [cpVect](#) **n**
- [cpFloat](#) **jAcc**

4.14.1 Detailed Description

A spring with a damper.

The documentation for this struct was generated from the following file:

- cpConstraint_private.h

4.15 cpGearJoint Struct Reference

Maintains a specific angular velocity between the two bodies.

```
#include <cpConstraint_private.h>
```

Public Attributes

- [cpConstraint](#) **constraint**
- [cpFloat](#) **phase**
- [cpFloat](#) **ratio**
- [cpFloat](#) **ratio_inv**
- [cpFloat](#) **iSum**
- [cpFloat](#) **bias**
- [cpFloat](#) **jAcc**

4.15.1 Detailed Description

Maintains a specific angular velocity between the two bodies.

The documentation for this struct was generated from the following file:

- cpConstraint_private.h

4.16 cpGrooveJoint Struct Reference

Similar to a pivot joint, but one of the anchors is a line segment that the pivot can slide in.

```
#include <cpConstraint_private.h>
```

Public Attributes

- [cpConstraint](#) **constraint**
- [cpVect](#) **grv_n**
- [cpVect](#) **grv_a**
- [cpVect](#) **grv_b**
- [cpVect](#) **anchorB**
- [cpVect](#) **grv_tn**
- [cpFloat](#) **clamp**
- [cpVect](#) **r1**
- [cpVect](#) **r2**
- [cpMat2x2](#) **k**
- [cpVect](#) **jAcc**
- [cpVect](#) **bias**

4.16.1 Detailed Description

Similar to a pivot joint, but one of the anchors is a line segment that the pivot can slide in.

The documentation for this struct was generated from the following file:

- `cpConstraint_private.h`

4.17 `cpMat2x2` Struct Reference

2x2 matrix type used for tensors and such

```
#include <chipmunk_types.h>
```

Public Attributes

- `cpFloat a`
- `cpFloat b`
- `cpFloat c`
- `cpFloat d`

4.17.1 Detailed Description

2x2 matrix type used for tensors and such

The documentation for this struct was generated from the following file:

- `chipmunk_types.h`

4.18 `cpPinJoint` Struct Reference

The two anchor points are always the same distance apart.

```
#include <cpConstraint_private.h>
```

Public Attributes

- `cpConstraint constraint`
- `cpVect anchorA`
- `cpVect anchorB`
- `cpFloat dist`
- `cpVect r1`
- `cpVect r2`
- `cpVect n`
- `cpFloat nMass`
- `cpFloat jnAcc`
- `cpFloat bias`

4.18.1 Detailed Description

The two anchor points are always the same distance apart.

The documentation for this struct was generated from the following file:

- `cpConstraint_private.h`

4.19 cpPivotJoint Struct Reference

Pivot joints hold two points on two bodies together allowing them to rotate freely around the pivot.

```
#include <cpConstraint_private.h>
```

Public Attributes

- [cpConstraint](#) **constraint**
- [cpVect](#) **anchorA**
- [cpVect](#) **anchorB**
- [cpVect](#) **r1**
- [cpVect](#) **r2**
- [cpMat2x2](#) **k**
- [cpVect](#) **jAcc**
- [cpVect](#) **bias**

4.19.1 Detailed Description

Pivot joints hold two points on two bodies together allowing them to rotate freely around the pivot.

The documentation for this struct was generated from the following file:

- cpConstraint_private.h

4.20 cpPointQueryInfo Struct Reference

Point query info struct.

```
#include <cpShape.h>
```

Public Attributes

- const [cpShape](#) * **shape**
The nearest shape, NULL if no shape was within range.
- [cpVect](#) **point**
The closest point on the shape's surface. (in world space coordinates)
- [cpFloat](#) **distance**
The distance to the point. The distance is negative if the point is inside the shape.
- [cpVect](#) **gradient**
The gradient of the signed distance function.

4.20.1 Detailed Description

Point query info struct.

4.20.2 Member Data Documentation

4.20.2.1 `cpVect` `cpPointQueryInfo::gradient`

The gradient of the signed distance function.

The value should be similar to `info.p/info.d`, but accurate even for very small values of `info.d`.

The documentation for this struct was generated from the following file:

- `cpShape.h`

4.21 `cpPolyline` Struct Reference

Public Attributes

- `int` **count**
- `int` **capacity**
- `cpVect` **verts** []

The documentation for this struct was generated from the following file:

- `cpPolyline.h`

4.22 `cpPolylineSet` Struct Reference

Polyline sets are collections of polylines, generally built by `cpMarchSoft()` or `cpMarchHard()`.

```
#include <cpPolyline.h>
```

Public Attributes

- `int` **count**
- `int` **capacity**
- `cpPolyline` ** **lines**

4.22.1 Detailed Description

Polyline sets are collections of polylines, generally built by `cpMarchSoft()` or `cpMarchHard()`.

The documentation for this struct was generated from the following file:

- `cpPolyline.h`

4.23 `cpPolyShape` Struct Reference

A convex polygon shape.

```
#include <cpShape_private.h>
```


Public Attributes

- [cpShape](#) **shape**
- [cpFloat](#) **r**
- int **count**
- struct [cpSplittingPlane](#) * **planes**
- struct [cpSplittingPlane](#) **_planes** [2 *CP_POLY_SHAPE_INLINE_ALLOC]

4.23.1 Detailed Description

A convex polygon shape.

The documentation for this struct was generated from the following file:

- cpShape_private.h

4.24 cpPostStepCallback Struct Reference

Public Attributes

- [cpPostStepFunc](#) **func**
- void * **key**
- void * **data**

The documentation for this struct was generated from the following file:

- cpSpace_private.h

4.25 cpRatchetJoint Struct Reference

Create rotary ratches similar to a socket wrench.

```
#include <cpConstraint_private.h>
```

Public Attributes

- [cpConstraint](#) **constraint**
- [cpFloat](#) **angle**
- [cpFloat](#) **phase**
- [cpFloat](#) **ratchet**
- [cpFloat](#) **iSum**
- [cpFloat](#) **bias**
- [cpFloat](#) **jAcc**

4.25.1 Detailed Description

Create rotary ratches similar to a socket wrench.

The documentation for this struct was generated from the following file:

- cpConstraint_private.h

4.26 cpRotaryLimitJoint Struct Reference

Constrains the bodies' orientations to be within a certain angle of each other.

```
#include <cpConstraint_private.h>
```

Public Attributes

- [cpConstraint](#) **constraint**
- [cpFloat](#) **min**
- [cpFloat](#) **max**
- [cpFloat](#) **iSum**
- [cpFloat](#) **bias**
- [cpFloat](#) **jAcc**

4.26.1 Detailed Description

Constrains the bodies' orientations to be within a certain angle of each other.

The documentation for this struct was generated from the following file:

- cpConstraint_private.h

4.27 cpSegmentQueryInfo Struct Reference

Segment query info struct.

```
#include <cpShape.h>
```

Public Attributes

- const [cpShape](#) * **shape**
The shape that was hit, or NULL if no collision occurred.
- [cpVect](#) **point**
The point of impact.
- [cpVect](#) **normal**
The normal of the surface hit.
- [cpFloat](#) **alpha**
The normalized distance along the query segment in the range [0, 1].

4.27.1 Detailed Description

Segment query info struct.

The documentation for this struct was generated from the following file:

- cpShape.h

4.28 cpSegmentShape Struct Reference

A beveled (rounded) segment shape.

```
#include <cpShape_private.h>
```

Public Attributes

- [cpShape](#) **shape**
- [cpVect](#) **a**
- [cpVect](#) **b**
- [cpVect](#) **n**
- [cpVect](#) **ta**
- [cpVect](#) **tb**
- [cpVect](#) **tn**
- [cpFloat](#) **r**
- [cpVect](#) **a_tangent**
- [cpVect](#) **b_tangent**

4.28.1 Detailed Description

A beveled (rounded) segment shape.

The documentation for this struct was generated from the following file:

- `cpShape_private.h`

4.29 cpShape Struct Reference

The [cpShape](#) struct defines the shape of a rigid body.

```
#include <cpShape_private.h>
```

Public Attributes

- const [cpShapeClass](#) * **klass**
- [cpSpace](#) * **space**
- [cpBody](#) * **body**
- struct [cpShapeMassInfo](#) **massInfo**
- [cpBB](#) **bb**
- [cpBool](#) **sensor**
- [cpFloat](#) **e**
- [cpFloat](#) **u**
- [cpVect](#) **surfaceV**
- [cpDataPointer](#) **userData**
- [cpCollisionType](#) **type**
- [cpShapeFilter](#) **filter**
- [cpShape](#) * **next**
- [cpShape](#) * **prev**
- [cpHashValue](#) **hashid**

4.29.1 Detailed Description

The [cpShape](#) struct defines the shape of a rigid body.

The documentation for this struct was generated from the following file:

- `cpShape_private.h`

4.30 cpShapeClass Struct Reference

Struct that holds function callback pointers for shapes.

```
#include <cpShape_private.h>
```

Public Attributes

- cpShapeType **type**
- cpShapeCacheDataImpl **cacheData**
- cpShapeDestroyImpl **destroy**
- cpShapePointQueryImpl **pointQuery**
- cpShapeSegmentQueryImpl **segmentQuery**

4.30.1 Detailed Description

Struct that holds function callback pointers for shapes.

The documentation for this struct was generated from the following file:

- cpShape_private.h

4.31 cpShapeFilter Struct Reference

Fast collision filtering type that is used to determine if two objects collide before calling collision or query callbacks.

```
#include <cpShape.h>
```

Public Attributes

- [cpGroup group](#)
Two objects with the same non-zero group value do not collide.
- [cpBitmask categories](#)
A bitmask of user definable categories that this object belongs to.
- [cpBitmask mask](#)
A bitmask of user definable category types that this object object collides with.

4.31.1 Detailed Description

Fast collision filtering type that is used to determine if two objects collide before calling collision or query callbacks.

4.31.2 Member Data Documentation

4.31.2.1 cpBitmask cpShapeFilter::categories

A bitmask of user definable categories that this object belongs to.

The category/mask combinations of both objects in a collision must agree for a collision to occur.

4.31.2.2 cpGroup cpShapeFilter::group

Two objects with the same non-zero group value do not collide.

This is generally used to group objects in a composite object together to disable self collisions.

4.31.2.3 cpBitmask cpShapeFilter::mask

A bitmask of user definable category types that this object object collides with.

The category/mask combinations of both objects in a collision must agree for a collision to occur.

The documentation for this struct was generated from the following file:

- cpShape.h

4.32 cpShapeMassInfo Struct Reference

Struct that holds information about the mass of the shape.

```
#include <cpShape_private.h>
```

Public Attributes

- cpFloat m
- cpFloat i
- cpVect cog
- cpFloat area

4.32.1 Detailed Description

Struct that holds information about the mass of the shape.

The documentation for this struct was generated from the following file:

- cpShape_private.h

4.33 cpSimpleMotor Struct Reference

Maintains a specific angular relative velocity between two objects.

```
#include <cpConstraint_private.h>
```

Public Attributes

- cpConstraint constraint
- cpFloat rate
- cpFloat iSum
- cpFloat jAcc

4.33.1 Detailed Description

Maintains a specific angular relative velocity between two objects.

The documentation for this struct was generated from the following file:

- cpConstraint_private.h

4.34 cpSlideJoint Struct Reference

Slide joints hold the distance between points on two bodies between a minimum and a maximum.

```
#include <cpConstraint_private.h>
```

Public Attributes

- [cpConstraint](#) **constraint**
- [cpVect](#) **anchorA**
- [cpVect](#) **anchorB**
- [cpFloat](#) **min**
- [cpFloat](#) **max**
- [cpVect](#) **r1**
- [cpVect](#) **r2**
- [cpVect](#) **n**
- [cpFloat](#) **nMass**
- [cpFloat](#) **jnAcc**
- [cpFloat](#) **bias**

4.34.1 Detailed Description

Slide joints hold the distance between points on two bodies between a minimum and a maximum.

The documentation for this struct was generated from the following file:

- cpConstraint_private.h

4.35 cpSpace Struct Reference

Containers for simulating objects in Chipmunk.

```
#include <cpSpace_private.h>
```

Public Attributes

- int **iterations**
- [cpVect](#) **gravity**
- [cpFloat](#) **damping**
- [cpFloat](#) **idleSpeedThreshold**
- [cpFloat](#) **sleepTimeThreshold**
- [cpFloat](#) **collisionSlop**
- [cpFloat](#) **collisionBias**
- [cpTimestamp](#) **collisionPersistence**
- [cpDataPointer](#) **userData**
- [cpTimestamp](#) **stamp**
- [cpFloat](#) **curr_dt**
- [cpArray](#) * **dynamicBodies**
- [cpArray](#) * **staticBodies**
- [cpArray](#) * **rousedBodies**
- [cpArray](#) * **sleepingComponents**
- [cpHashValue](#) **shapelDCounter**
- [cpSpatialIndex](#) * **staticShapes**

- cpSpatialIndex * **dynamicShapes**
- cpArray * **constraints**
- cpArray * **arbiters**
- cpContactBufferHeader * **contactBuffersHead**
- cpHashSet * **cachedArbiters**
- cpArray * **pooledArbiters**
- cpArray * **allocatedBuffers**
- unsigned int **locked**
- cpBool **usesWildcards**
- cpHashSet * **collisionHandlers**
- cpCollisionHandler **defaultHandler**
- cpBool **skipPostStep**
- cpArray * **postStepCallbacks**
- cpBody * **staticBody**
- cpBody **_staticBody**

4.35.1 Detailed Description

Containers for simulating objects in Chipmunk.

The documentation for this struct was generated from the following file:

- cpSpace_private.h

4.36 cpSpaceDebugColor Struct Reference

Color type to use with the space debug drawing API.

```
#include <cpSpace.h>
```

Public Attributes

- float **r**
- float **g**
- float **b**
- float **a**

4.36.1 Detailed Description

Color type to use with the space debug drawing API.

The documentation for this struct was generated from the following file:

- cpSpace.h

4.37 cpSpaceDebugDrawOptions Struct Reference

Struct used with [cpSpaceDebugDraw\(\)](#) containing drawing callbacks and other drawing settings.

```
#include <cpSpace.h>
```

Public Attributes

- [cpSpaceDebugDrawCircleImpl drawCircle](#)
Function that will be invoked to draw circles.
- [cpSpaceDebugDrawSegmentImpl drawSegment](#)
Function that will be invoked to draw line segments.
- [cpSpaceDebugDrawFatSegmentImpl drawFatSegment](#)
Function that will be invoked to draw thick line segments.
- [cpSpaceDebugDrawPolygonImpl drawPolygon](#)
Function that will be invoked to draw convex polygons.
- [cpSpaceDebugDrawDotImpl drawDot](#)
Function that will be invoked to draw dots.
- [cpSpaceDebugDrawFlags flags](#)
Flags that request which things to draw (collision shapes, constraints, contact points).
- [cpSpaceDebugColor shapeOutlineColor](#)
Outline color passed to the drawing function.
- [cpSpaceDebugDrawColorForShapeImpl colorForShape](#)
Function that decides what fill color to draw shapes using.
- [cpSpaceDebugColor constraintColor](#)
Color passed to drawing functions for constraints.
- [cpSpaceDebugColor collisionPointColor](#)
Color passed to drawing functions for collision points.
- [cpDataPointer data](#)
User defined context pointer passed to all of the callback functions as the 'data' argument.

4.37.1 Detailed Description

Struct used with [cpSpaceDebugDraw\(\)](#) containing drawing callbacks and other drawing settings.

The documentation for this struct was generated from the following file:

- [cpSpace.h](#)

4.38 cpSpatialIndexClass Struct Reference

Used to accelerate collision detection.

```
#include <cpSpatialIndex.h>
```

Public Attributes

- [cpSpatialIndexDestroyImpl destroy](#)
- [cpSpatialIndexCountImpl count](#)
- [cpSpatialIndexEachImpl each](#)
- [cpSpatialIndexContainsImpl contains](#)
- [cpSpatialIndexInsertImpl insert](#)
- [cpSpatialIndexRemoveImpl remove](#)
- [cpSpatialIndexReindexImpl reindex](#)
- [cpSpatialIndexReindexObjectImpl reindexObject](#)
- [cpSpatialIndexReindexQueryImpl reindexQuery](#)
- [cpSpatialIndexQueryImpl query](#)
- [cpSpatialIndexSegmentQueryImpl segmentQuery](#)

4.38.1 Detailed Description

Used to accelerate collision detection.

The documentation for this struct was generated from the following file:

- cpSpatialIndex.h

4.39 cpSplittingPlane Struct Reference

Splitting plane.

```
#include <cpShape_private.h>
```

Public Attributes

- [cpVect](#) **v0**
- [cpVect](#) **n**

4.39.1 Detailed Description

Splitting plane.

The documentation for this struct was generated from the following file:

- cpShape_private.h

4.40 cpTransform Struct Reference

Column major affine transform.

```
#include <chipmunk_types.h>
```

Public Attributes

- [cpFloat](#) **a**
- [cpFloat](#) **b**
- [cpFloat](#) **c**
- [cpFloat](#) **d**
- [cpFloat](#) **tx**
- [cpFloat](#) **ty**

4.40.1 Detailed Description

Column major affine transform.

The documentation for this struct was generated from the following file:

- chipmunk_types.h

4.41 cpVect Struct Reference

Chipmunk's 2D vector type.

```
#include <chipmunk_types.h>
```

Public Attributes

- [cpFloat](#) **x**
- [cpFloat](#) **y**

4.41.1 Detailed Description

Chipmunk's 2D vector type.

The documentation for this struct was generated from the following file:

- chipmunk_types.h

Index

Basic Types, 8

cpCollisionID, 9

cpFloat, 9

CP_BODY_TYPE_DYNAMIC

cpBody, 18

CP_BODY_TYPE_KINEMATIC

cpBody, 18

CP_BODY_TYPE_STATIC

cpBody, 18

CP_CONVEX_HULL

Misc, 6

categories

cpShapeFilter, 68

Chipmunk Unsafe Shape Operations, 10

cpArbiter, 11, 53

cpArbiterCallWildcardBeginA, 12

cpArbiterCallWildcardBeginB, 12

cpArbiterCallWildcardPreSolveA, 13

cpArbiterCallWildcardPreSolveB, 13

cpArbiterGetBodies, 13

cpArbiterGetShapes, 13

cpArbiterIgnore, 13

cpArbiterSetContactPointSet, 13

cpArbiterSetUserData, 13

cpArbiterTotalImpulse, 13

cpArbiterTotalKE, 14

cpArbiterCallWildcardBeginA

cpArbiter, 12

cpArbiterCallWildcardBeginB

cpArbiter, 12

cpArbiterCallWildcardPreSolveA

cpArbiter, 13

cpArbiterCallWildcardPreSolveB

cpArbiter, 13

cpArbiterGetBodies

cpArbiter, 13

cpArbiterGetShapes

cpArbiter, 13

cpArbiterIgnore

cpArbiter, 13

cpArbiterSetContactPointSet

cpArbiter, 13

cpArbiterSetUserData

cpArbiter, 13

cpArbiterThread, 53

cpArbiterTotalImpulse

cpArbiter, 13

cpArbiterTotalKE

cpArbiter, 14

cpAreaForCircle

Misc, 7

cpAreaForPoly

Misc, 7

cpArray, 54

cpBB, 15, 54

cpBBTreeVelocityFunc

cpSpatialIndex, 48

cpBody, 16, 55

CP_BODY_TYPE_DYNAMIC, 18

CP_BODY_TYPE_KINEMATIC, 18

CP_BODY_TYPE_STATIC, 18

cpBodySetPositionUpdateFunc, 19

cpBodyType, 18

cpBodySetPositionUpdateFunc

cpBody, 19

cpBodyType

cpBody, 18

cpCircleShape, 35, 55

cpCollisionBeginFunc

cpSpace, 42

cpCollisionHandler, 56

postSolveFunc, 56

preSolveFunc, 56

typeA, 56

typeB, 57

cpCollisionID

Basic Types, 9

cpCollisionInfo, 57

cpCollisionPreSolveFunc

cpSpace, 42

cpConstraint, 20, 57

cpConstraintSetErrorBias, 21

cpConstraintClass, 58

cpConstraintSetErrorBias

cpConstraint, 21

cpContact, 58

cpContactPointSet, 59

distance, 59

cpConvexHull

Misc, 7

cpDampedRotarySpring, 22, 60

cpDampedSpring, 23, 60

cpFloat

Basic Types, 9

cpGearJoint, 24, 61

cpGrooveJoint, 25, 61

cpMat2x2, 52, 62

cpMomentForCircle

- Misc, [7](#)
- cpMomentForSegment
 - Misc, [7](#)
- cpPinJoint, [26](#), [62](#)
- cpPivotJoint, [27](#), [63](#)
- cpPointQueryInfo, [63](#)
 - gradient, [64](#)
- cpPolyShape, [28](#), [64](#)
 - cpPolyShapeInit, [28](#)
 - cpPolyShapeInitRaw, [28](#)
 - cpPolyShapeNew, [29](#)
 - cpPolyShapeNewRaw, [29](#)
- cpPolyShapeInit
 - cpPolyShape, [28](#)
- cpPolyShapeInitRaw
 - cpPolyShape, [28](#)
- cpPolyShapeNew
 - cpPolyShape, [29](#)
- cpPolyShapeNewRaw
 - cpPolyShape, [29](#)
- cpPolyline, [64](#)
- cpPolylineSet, [64](#)
- cpPostStepCallback, [65](#)
- cpRatchetJoint, [30](#), [65](#)
- cpRotaryLimitJoint, [31](#), [66](#)
- cpSegmentQueryInfo, [66](#)
- cpSegmentShape, [36](#), [66](#)
- cpShape, [32](#), [67](#)
 - cpShapePointQuery, [34](#)
 - cpShapeSetBody, [34](#)
- cpShapeClass, [68](#)
- cpShapeFilter, [68](#)
 - categories, [68](#)
 - group, [68](#)
 - mask, [68](#)
- cpShapeMassInfo, [69](#)
- cpShapePointQuery
 - cpShape, [34](#)
- cpShapeSetBody
 - cpShape, [34](#)
- cpSimpleMotor, [37](#), [69](#)
- cpSlideJoint, [38](#), [70](#)
- cpSpace, [39](#), [70](#)
 - cpCollisionBeginFunc, [42](#)
 - cpCollisionPreSolveFunc, [42](#)
 - cpSpaceAddCollisionHandler, [43](#)
 - cpSpaceAddPostStepCallback, [43](#)
 - cpSpaceAddShape, [43](#)
 - cpSpaceBBQuery, [43](#)
 - cpSpaceGetCollisionBias, [43](#)
 - cpSpaceGetCollisionPersistence, [43](#)
 - cpSpaceGetCollisionSlop, [43](#)
 - cpSpaceGetCurrentTimeStep, [43](#)
 - cpSpaceGetDamping, [44](#)
 - cpSpaceGetIdleSpeedThreshold, [44](#)
 - cpSpaceGetSleepTimeThreshold, [44](#)
 - cpSpaceGetStaticBody, [44](#)
 - cpSpaceGetUserData, [44](#)
 - cpSpaceSetCollisionBias, [44](#)
 - cpSpaceSetCollisionPersistence, [44](#)
 - cpSpaceSetCollisionSlop, [44](#)
 - cpSpaceSetDamping, [45](#)
 - cpSpaceSetIdleSpeedThreshold, [45](#)
 - cpSpaceSetSleepTimeThreshold, [45](#)
 - cpSpaceSetUserData, [45](#)
 - cpSpatialIndex, [46](#)
 - cpBBTreeVelocityFunc, [48](#)
 - cpSpaceHashResize, [48](#)
 - cpSpatialIndexBBFunc, [48](#)
 - cpSpatialIndexContains, [48](#)
 - cpSpatialIndexInsert, [48](#)
 - cpSpatialIndexReindexQuery, [48](#)
- cpSpaceSetCollisionBias, [44](#)
- cpSpaceSetCollisionPersistence, [44](#)
- cpSpaceSetCollisionSlop, [44](#)
- cpSpaceSetDamping, [45](#)
- cpSpaceSetIdleSpeedThreshold, [45](#)
- cpSpaceSetSleepTimeThreshold, [45](#)
- cpSpaceSetUserData, [45](#)
- cpSpaceAddCollisionHandler
 - cpSpace, [43](#)
- cpSpaceAddPostStepCallback
 - cpSpace, [43](#)
- cpSpaceAddShape
 - cpSpace, [43](#)
- cpSpaceBBQuery
 - cpSpace, [43](#)
- cpSpaceDebugColor, [71](#)
- cpSpaceDebugDrawOptions, [71](#)
- cpSpaceGetCollisionBias
 - cpSpace, [43](#)
- cpSpaceGetCollisionPersistence
 - cpSpace, [43](#)
- cpSpaceGetCollisionSlop
 - cpSpace, [43](#)
- cpSpaceGetCurrentTimeStep
 - cpSpace, [43](#)
- cpSpaceGetDamping
 - cpSpace, [44](#)
- cpSpaceGetIdleSpeedThreshold
 - cpSpace, [44](#)
- cpSpaceGetSleepTimeThreshold
 - cpSpace, [44](#)
- cpSpaceGetStaticBody
 - cpSpace, [44](#)
- cpSpaceGetUserData
 - cpSpace, [44](#)
- cpSpaceHashResize
 - cpSpatialIndex, [48](#)
- cpSpaceSetCollisionBias
 - cpSpace, [44](#)
- cpSpaceSetCollisionPersistence
 - cpSpace, [44](#)
- cpSpaceSetCollisionSlop
 - cpSpace, [44](#)
- cpSpaceSetDamping
 - cpSpace, [45](#)
- cpSpaceSetIdleSpeedThreshold
 - cpSpace, [45](#)
- cpSpaceSetSleepTimeThreshold
 - cpSpace, [45](#)
- cpSpaceSetUserData
 - cpSpace, [45](#)
- cpSpatialIndex, [46](#)
 - cpBBTreeVelocityFunc, [48](#)
 - cpSpaceHashResize, [48](#)
 - cpSpatialIndexBBFunc, [48](#)
 - cpSpatialIndexContains, [48](#)
 - cpSpatialIndexInsert, [48](#)
 - cpSpatialIndexReindexQuery, [48](#)

- cpSpatialIndexRemove, [48](#)
- cpSpatialIndexBBFunc
 - cpSpatialIndex, [48](#)
- cpSpatialIndexClass, [72](#)
- cpSpatialIndexContains
 - cpSpatialIndex, [48](#)
- cpSpatialIndexInsert
 - cpSpatialIndex, [48](#)
- cpSpatialIndexReindexQuery
 - cpSpatialIndex, [48](#)
- cpSpatialIndexRemove
 - cpSpatialIndex, [48](#)
- cpSplittingPlane, [73](#)
- cpTransform, [73](#)
- cpVect, [50](#), [74](#)
 - cpvcross, [51](#)
- cpvcross
 - cpVect, [51](#)
- distance
 - cpContactPointSet, [59](#)
- gradient
 - cpPointQueryInfo, [64](#)
- group
 - cpShapeFilter, [68](#)
- mask
 - cpShapeFilter, [68](#)
- Misc, [5](#)
 - CP_CONVEX_HULL, [6](#)
 - cpAreaForCircle, [7](#)
 - cpAreaForPoly, [7](#)
 - cpConvexHull, [7](#)
 - cpMomentForCircle, [7](#)
 - cpMomentForSegment, [7](#)
- postSolveFunc
 - cpCollisionHandler, [56](#)
- preSolveFunc
 - cpCollisionHandler, [56](#)
- typeA
 - cpCollisionHandler, [56](#)
- typeB
 - cpCollisionHandler, [57](#)