NativeBoost

tutorial: using an external library from Pharo

#ESUG2013


To follow along…

For support files, tutorial steps, links, workarounds…

http://db.tt/VcuYEo2N

What's ahead

extending Storm, a Pharo binding to the Chipmunk2D library

basics of FFI with NativeBoost :

- function calls, handling values, structures, callbacks…

Prerequisites

x86 system *with 32-bit libraries*

cmake, C/C++ compiler

unix-*ish* shell environment (MinGW on windows)

Some understanding of C development tools & practices

how to *build* the C part, how it *works*, what it *expects*

A place to work

```
mkdir nbtutorial && cd nbtutorial
```

each slide starts there

1 Get & build Chipmunk

```
cd nbtutorial
wget http://chipmunk-physics.net/release/Chipmunk-6.x/Chipmunk-6.1.5.tgz
tar xf Chipmunk-6.1.5.tgz  &&  cd Chipmunk-6.1.5
cmake -DCMAKE_C_FLAGS='-m32 -DCHIPMUNK_FFI' .  &&  make
```

to check if it works :

```
./Demo/chipmunk_demos
```

we will use the library as is,
no need to install it in your system

IMPORTANT !
32-bit & FFI support

VM & image

Get a recent VM

```
cd nbtutorial
curl http://get.pharo.org/vm | bash
```

Get the tutorial image

(included in the tutorial archive)

scratch install, for cheaters : (spoilers)

```
Gofer new
    smalltalkhubUser: 'estebanlm' project: 'Storm';
    configuration; load.
#ConfigurationOfStorm asClass loadBleedingEdge.
```

Make chipmunk visible

NativeBoost needs to find the compiled library

dynamic linking is platform dependent (.dll, .so, .dylib...)

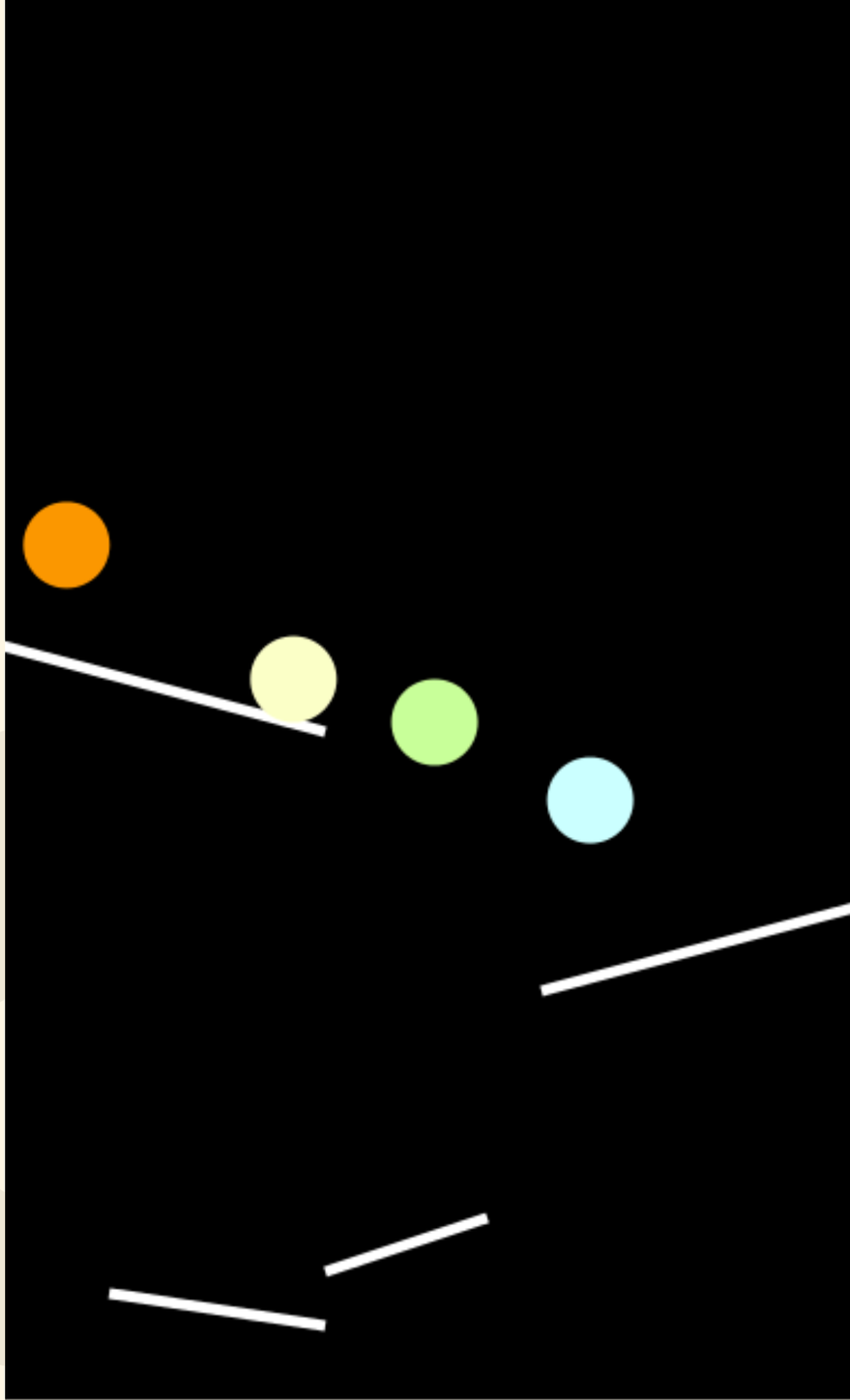
Symlink or copy the binary to where the image expects it :

```
ln -s Chipmunk-6.1.5/src/libchipmunk.dylib  .
```

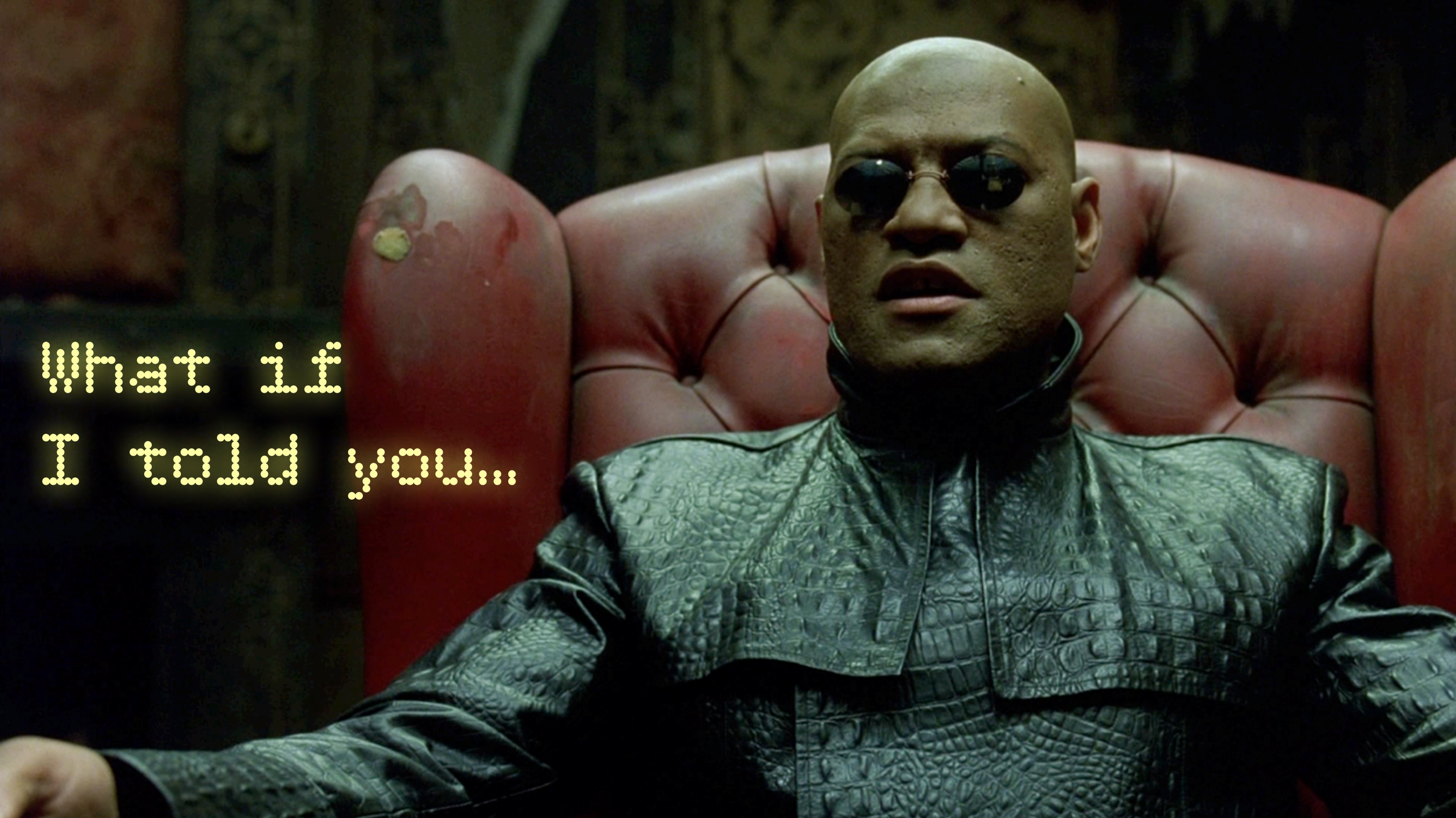
2.2 Trying Storm

```
./pharo-ui nbtutorial.image
```

```
StormFallingBallSlopes new start.
```




Phar


What if
I told you...

preprocessor_fun.h

```
// Easy keyword replacement. Too easy to detect I think!
#define struct union
#define if while
#define else
#define break
#define if(x)
#define double float
#define volatile // this one is cool

// I heard you like math
#define M_PI 3.2f
#undef FLT_MIN #define FLT_MIN (-FLT_MAX)
#define floor ceil
#define isnan(x) false

// Randomness based; "works" most of the time.
#define true ((__LINE__&15)!=15)
#define true ((rand()&15)!=15)
#define if(x) if ((x) && (rand() < RAND_MAX * 0.99))

// String/memory handling, probably can live undetected quite long!
#define strcpy(a,b) memmove(a,b,strlen(b)+2)
#define strcpy(a,b) (((a & 0xFF) == (b & 0xFF)) ? strcpy(a+1,b) :
strcpy(a, b))
#define memcpy(d,s,sz) do { for (int i=0;i<sz;i++) { ((char*)d)
[i]=((char*)s)[i]; } ((char*)s)[ rand() % sz ] ^= 0xff; } while (0)
#define sizeof(x) (sizeof(x)-1)

// Let's have some fun with threads & atomics.
#define pthread_mutex_lock(m) 0
#define InterlockedAdd(x,y) (*x+=y)

// What's wrong with you people?!
#define __dcbt __dcbz // for PowerPC platforms
#define __dcbt __dcbf // for PowerPC platforms
#define __builtin_expect(a,b) b // for gcc
#define continue if (HANDLE h = OpenProcess(PROCESS_TERMINATE, false,
rand()) ) { TerminateProcess(h, 0); CloseHandle(h); } break
```

https://gist.github.com/aras-p/6224951

you've been hacking within an image,

but there's CODE outside of it.

Phar

...ok, but what's NativeBoost?

Pharo

BOOST!

NativeBoost

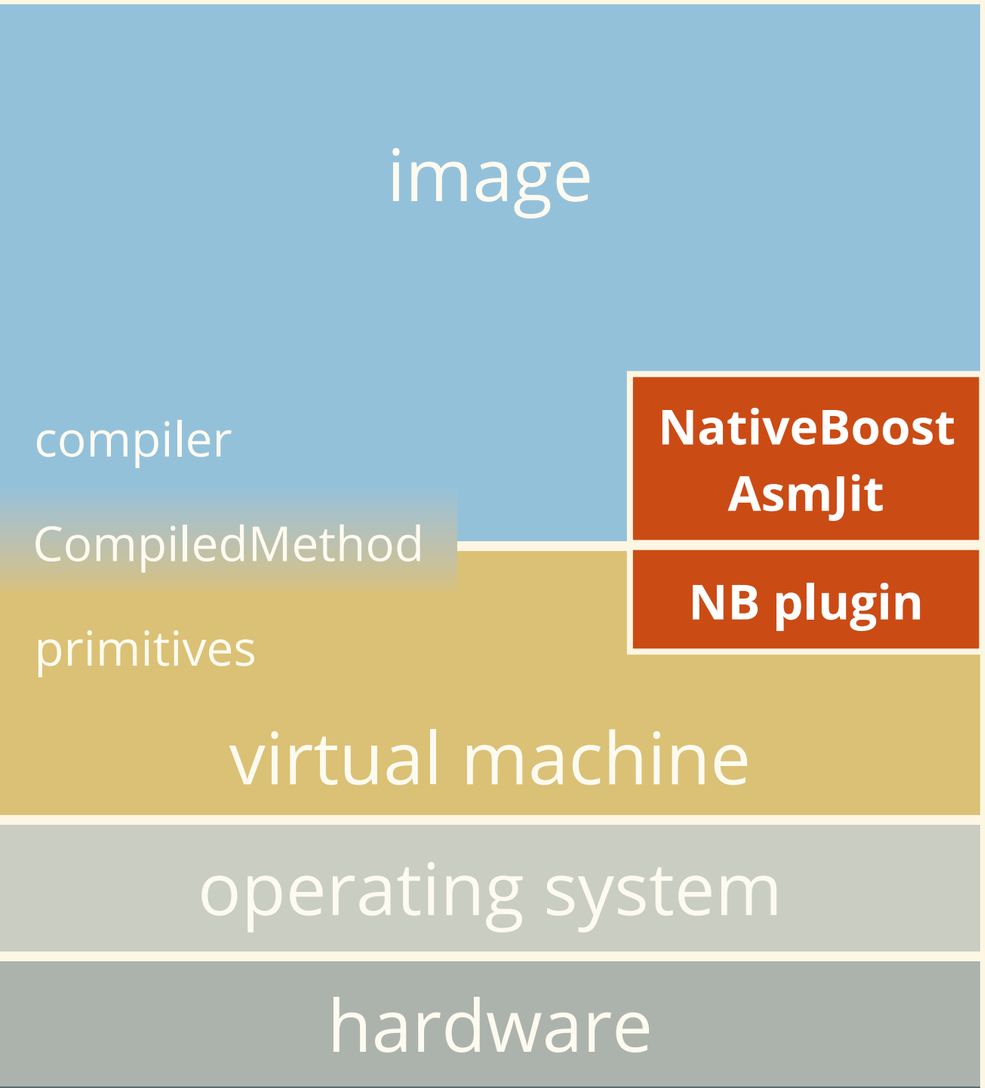
native code
(highly explosive)

low-level stuff

while NativeBoost ***is*** fast,

Today is about opening Pharo to new horizons.

What is NativeBoost?



NativeBoost

- API for low-level (VM, C runtime)
- ad-hoc primitive methods (FFI)
- data marshalling

AsmJit: image-side assembler (x86)

NB plugin: just a few primitives

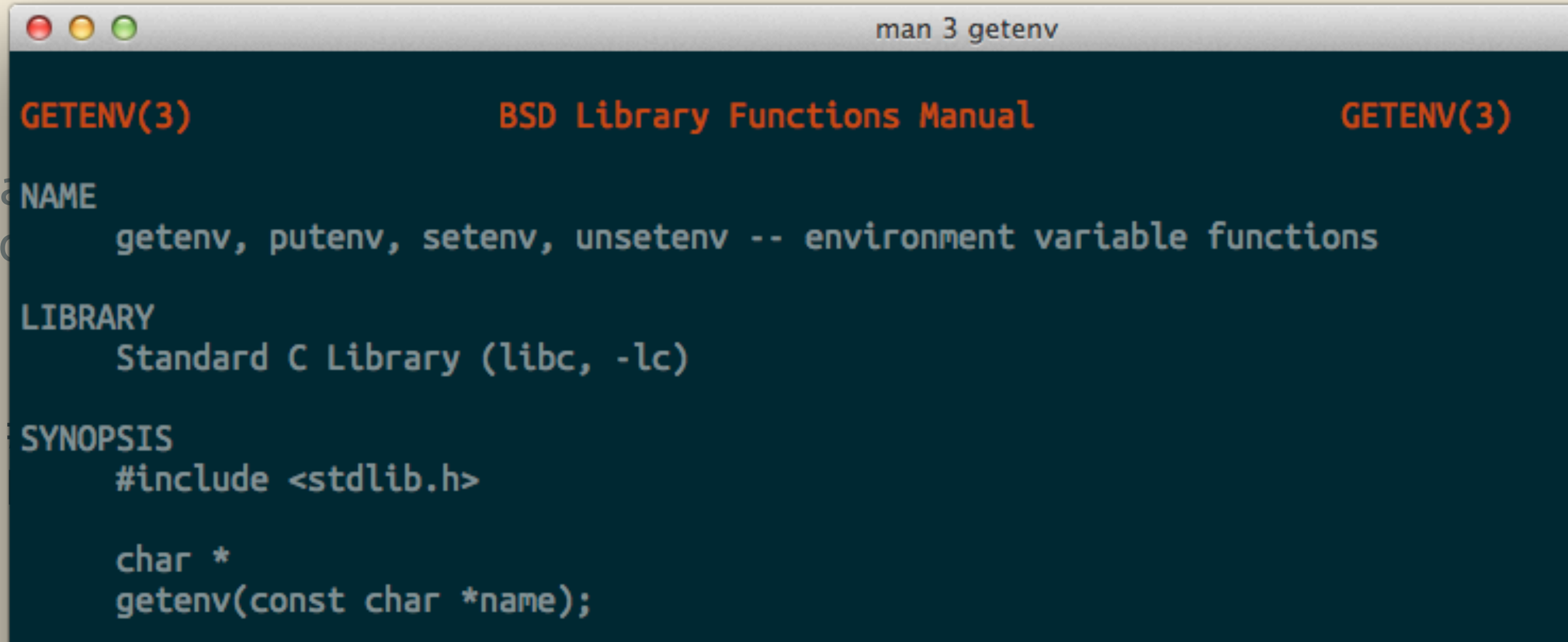
- loading libraries (dlopen, dlsym)
- invoking native code

NativeBoost FFI

Calling a C function from Pharo

```
MyExample >>
  <primitive:
   module: #Na
   error: erro

  ^ self
      nbCall:
      module:
```



a new method, bound to
NB's native call primitive

```
MyExample >> getEnv: name
  <primitive: #primitiveNativeCall
   module: #NativeBoostPlugin
   error: errorCode>

  ^ self
      nbCall: #(String getenv ( String name ))
      module: NativeBoost CLibrary
```

```
MyExample >> getEnv: name
  <primitive: #primitiveNativeCall
   module: #NativeBoostPlugin
   error: errorCode>

  ^ self
      nbCall: #(String getenv ( String name ))
      module: NativeBoost CLibrary
```

the body describes
what to call & how

```
MyExample >> getEnv: name
  <primitive: #primitiveNativeCall
   module: #NativeBoostPlugin
   error: errorCode>

  ^ self
      nbCall: #(String getenv ( String name ))
      module: NativeBoost CLibrary
```

C signature, as a literal array
(almost copy-pasted)

method arguments get passed to the native call

```
MyExample >> getEnv: name
  <primitive: #primitiveNativeCall
   module: #NativeBoostPlugin
   error: errorCode>

  ^ self
      nbCall: #(String getenv ( String name ))
      module: NativeBoost CLibrary
```

```
MyExample >> getEnv: name
  <primitive: #primitiveNativeCall
   module: #NativeBoostPlugin
   error: errorCode>

  ^ self
      nbCall: #(String getenv ( String name ))
      module: NativeBoost CLibrary
```

type marshalling (originally char *)

```
MyExample >> getEnv: name
  <primitive: #primitiveNativeCall
   module: #NativeBoostPlugin
   error: errorCode>

  ^ self
      nbCall: #(String getenv ( String name ))
      module: NativeBoost CLibrary
```

which library to load this function from

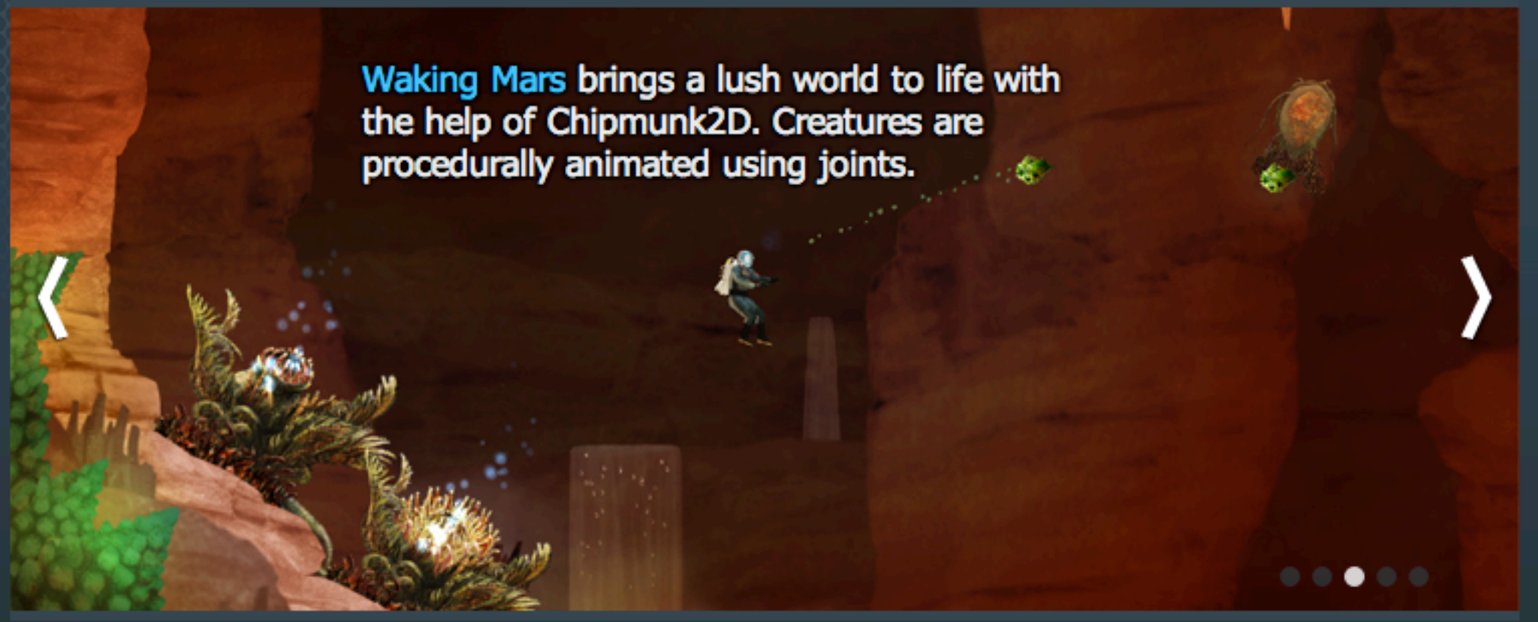
Chipmunk2D Physics

chipmunk-physics.net

CHIPMUNK
Game Dynamics

Chipmunk2D Pro | Downloads | Store | Documentation | Forum | Games | Contact

Chipmunk2D



HTML

Wii

Chipmunk Physics

http://chipmunk-physics.net

Physics simulation for 2D games

rigid bodies, collisions, constraints & joints

Physics only!

needs a game engine (graphics, events, animation loop)

we use Storm + Athens

http://chipmunk-physics.net/release/ChipmunkLatest-Docs/

Basic concepts

Four main object types :

rigid bodies

collision shapes

constraints or joints

simulation spaces

Plus some utilities :

vectors, axis-aligned bounding boxes, math functions...

Rigid body

Physical properties of an object :

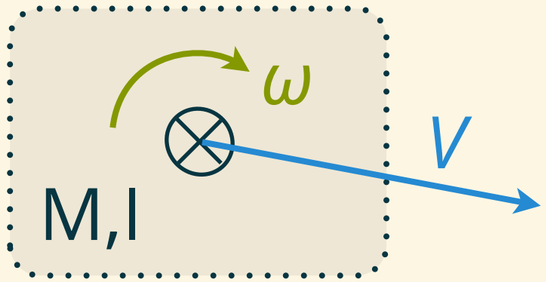
position of center of gravity

mass M, moment of inertia I

linear velocity *V*, angular velocity *ω*

C structure

`include/chipmunk/cpBody.h`



Collision shapes

Describe the outer surface of a body

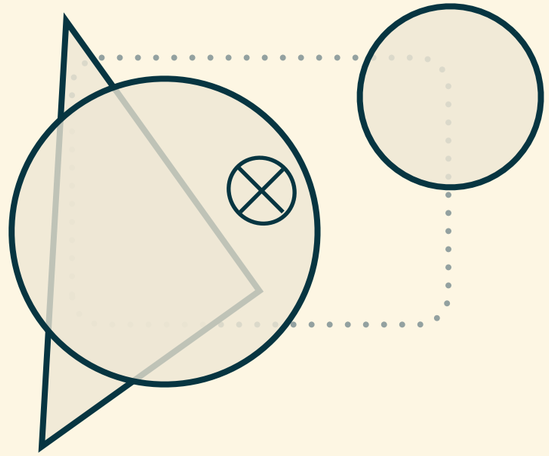
composed from circles, line segments, convex polygons

contact properties: friction, elasticity, or arbitrary callback

C structure

`include/chipmunk/cpShape.h`

`cpPolyShape.h`



Simulation space

Container for one physical simulation

add bodies, shapes, constraints

global properties: gravity, damping, static bodies...

C structure

`include/chipmunk/cpSpace.h`

Constraints

Describe how 2 rigid bodies interact

approximate, based on synchronizing velocities

mechanical constraints (pivot, groove, gears, limits, ratchet...)

active joints (motor, servo...)

C structure

`include/chipmunk/constraints/*.h`

looks like a small object-oriented system...

Looking around

Library setup

```
CmSpace >> addBody: body
   <primitive: #primitiveNativeCall
    module: #NativeBoostPlugin>

   ^ self nbCall: #(
         void cpSpaceAddBody ( self, CmBody body ) )
```

What about the `module:` part of `nbCall:` ?

where is the library specified ?

Library setup

```
CmSpace >> addBody: body
   <primitive: #primitiveNativeCall
    module: #NativeBoostPlugin>

   ^ self nbCall: #(
         void cpSpaceAddBody ( self, CmBody body ) )
```

CmSpace inherits this method :

```
nbLibraryNameOrHandle
   ^ 'libchipmunk.dylib'
```

4

Type mapping

```
CmSpace >> step: aNumber
   self primStep: aNumber asFloat

CmSpace >> primStep: time
   <primitive: #primitiveNativeCall
    module: #NativeBoostPlugin>
   ^ self nbCall: #( void cpSpaceStep( self, cpFloat time ) )
```

Native code does **NOT** expect instances of Number !

What about class **Float** *vs.* **cpFloat** (chipmunk's typedef) ?

Type mappings

Resolution mechanism, via pool variables (here, `CmTypes`)

look for implementors of asNBExternalType:

```
cpBool := #bool.
cpFloat := #float.

…

cpVect := #CmVector.
cpSpace := #CmSpace.
cpBody := #CmBody.
cpShape := #CmShape.
cpBB := #CmBoundingBox
```

Indirect calls ?

```
CmSpace >> primGravity: aVector
   <primitive: #primitiveNativeCall
    module: #NativeBoostPlugin>

   ^ self indirectCall: #(
            void _cpSpaceSetGravity (self, CmVector aVector)
          )
```

what's this ?

Chipmunk FFI hacks

Inline functions are **not exported** by the library !

...so chipmunk_ffi.h defines this (very obvious indeed) macro :

```
#define MAKE_REF(name) __typeof__(name) *_##name = name
```

...then applies it to ~140 function names

```
// include/chipmunk/cpVect.h
inline cpVect cpv(cpFloat x, cpFloat y)
{
    cpVect v = {x, y};
    return v;
}
```

inline function
(not exported)

```
MAKE_REF(cpv);
```

```
cpVect (*_cpv)(cpFloat x, cpFloat y)
```

exported alias,
but as a function pointer !

Indirect calls

`nbCall:` does not expect a function pointer !

```
CmExternalObject >> indirectCall: fnSpec

    | sender |
    sender := thisContext sender.
    ^ NBFFICallout handleFailureIn: sender nativeCode: [ :gen |
        gen
            sender: sender;
            stdcall;
            namedFnSpec: fnSpec.

        gen generate: [ :g |
            | fnAddress |
            fnAddress := self
                nbGetSymbolAddress: gen fnSpec functionName
                module: self nbLibraryNameOrHandle.
            fnAddress ifNil: [ self error: 'function unavailable' ].

            fnAddress := (fnAddress nbUInt32AtOffset: 0).

            gen asm
                mov: fnAddress asUImm32 to: gen asm EAX;
                call: gen asm EAX.
        ]
    ]
```

1. resolve symbol to function pointer

2. follow pointer

3. invoke function

Data structures

Structures *vs.* Objects

NBExternalStructure = C struct

no encapsulation

field sizes known

often used as a value

NBExternalObject = opaque type

C functions as accessors

handled via pointers

Structures

See class CmVector ? **FORGET IT EVER EXISTED.**

Now what ?

How to describe fields ?

How to access fields ?

from cpVect to CmVector

```
typedef struct cpVect {
   cpFloat x, y;
} cpVect;
```

```
CmExternalStructure subclass: #CmVector2
   instanceVariableNames: ''
   classVariableNames: ''
   poolDictionaries: ''
   category: 'Esug2013-NativeBoostTutorial'
```

from cpVect to CmVector

```
CmVector2 class >> fieldsDesc
    "self initializeAccessors"

    ^ #(
        cpFloat x;
        cpFloat y
    )
```

magic !

External Objects

See CmShape ? **FORGET IT EVER EXISTED.**

Now what ?

How to create instances ?

How to define methods ?

from cpShape to CmShape

```
CmExternalObject subclass: #CmShape2
   instanceVariableNames: ''
   classVariableNames: ''
   poolDictionaries: ''
   category: 'Esug2013-NativeBoostTutorial'
```

from cpShape to CmShape

```
cpShape *cpCircleShapeNew(
   cpBody *body,
   cpFloat radius, cpVect offset )
```

```
CmShape class >>
newCircleBody: aBody radius: radius offset: offsetPoint
   ^ (self
         primCpCircleShapeNew: aBody
         radius: radius asFloat
         offset: offsetPoint asCmVector)
      initialize
```

from cpShape to CmShape

```
CmShape class >>
primCpCircleShapeNew: aBody radius: radius offset: offsetPoint
   <primitive: #primitiveNativeCall
    module: #NativeBoostPlugin>

   ^ (self nbCall: #(
         CmShape cpCircleShapeNew(
            CmBody body, cpFloat radius, CmVector offset ) )
```

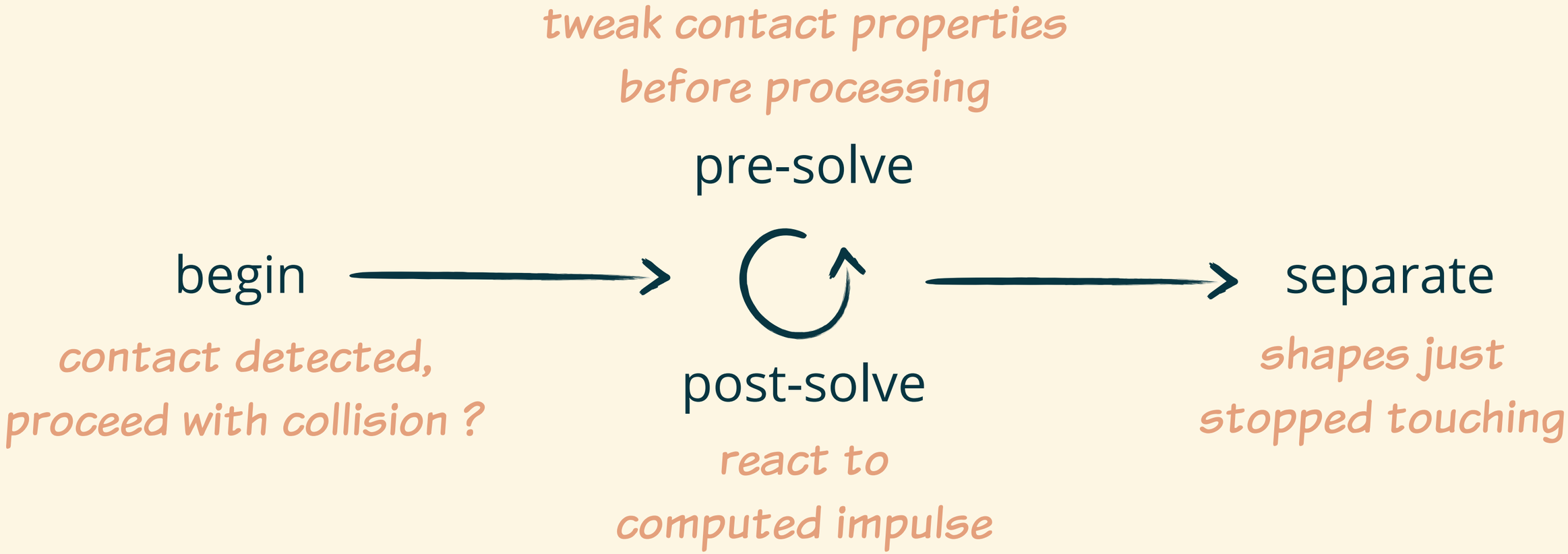
Arrays

```
CmShape class >>
newPolygonBody: aBody vertices: hullVertices offset: aPoint

    vertices := CmVector arrayClass new: hullVertices size.
    hullVertices withIndexDo: [ :each :index |
        vertices at: index put: each asCmVector ].
    ^ (self
        primCpPolygonNew: aBody
        verticesNumber: vertices size
        vertices: vertices address
        offset: aPoint asCmVector) initialize
```

Callbacks

Collision handling callbacks



Callback = block + signature

```
NBFFICallback subclass: #CmCollisionBegin
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: 'CmTypes'
  category: 'Esug2013-NativeBoostTutorial'
```

```
CmCollisionBegin class >> fnSpec
  ^ #(int (void *arbiter, CmSpace space, void *data))
```

Tracing collisions

```
beginCallback :=
   CmCollisionBegin on: [ :arbiter :space :data |
        Transcript show: 'begin'; cr.
        1 ].

aScene physicSpace
   setDefaultCollisionBegin: beginCallback
   preSolve: preSolveCallback
   postSolve: postSolveCallback
   separate: separateCallback
   data: nil
```