Gemini
Lua Scripting for iOS Games

James Norton
@jnorton

Lua Workshop 2012

Gemini

Gemini

- Provides Lua bindings for 2D game dev

Gemini

- Provides Lua bindings for 2D game dev
- Allows games to be coded entirely in Lua

Gemini

- Provides Lua bindings for 2D game dev
- Allows games to be coded entirely in Lua
- Consists of a set of Lua C libraries plus Objective C project code and templates

Gemini

- Provides Lua bindings for 2D game dev
- Allows games to be coded entirely in Lua
- Consists of a set of Lua C libraries plus Objective C project code and templates
- Similar API to the Corona™ SDK

Gemini

- Provides Lua bindings for 2D game dev
- Allows games to be coded entirely in Lua
- Consists of a set of Lua C libraries plus Objective C project code and templates
- Similar API to the Corona™ SDK
- Open source with MIT License

Why use scripting?

Why use scripting?

- Provides higher abstraction that can increase productivity.

Why use scripting?

- Provides higher abstraction that can increase productivity.
- Level designers and non-programmers can work with it.

Why use scripting?

- Provides higher abstraction that can increase productivity.
- Level designers and non-programmers can work with it.
- Functionality can be changed without recompiling during development.

Features

Features

- 2D Layer based rendering

Features

- 2D Layer based rendering
- Event Driven

Features

- 2D Layer based rendering
- Event Driven
- Graphics elements

Features

- 2D Layer based rendering
- Event Driven
- Graphics elements
- Physics

Features

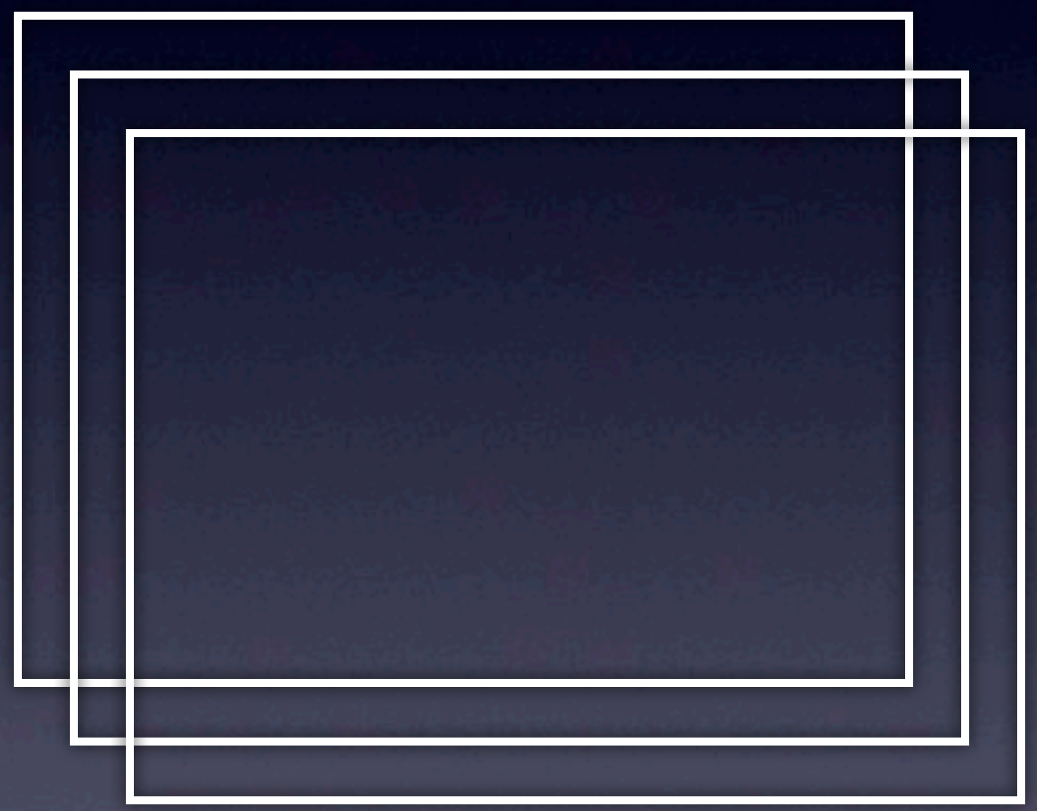
- 2D Layer based rendering
- Event Driven
- Graphics elements
- Physics
- Sound

Features

- 2D Layer based rendering
- Event Driven
- Graphics elements
- Physics
- Sound
- Scene Management

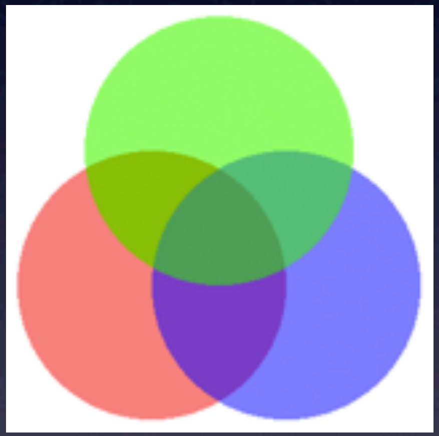
Layers

- Provide depth
- Parallax effect
- Blending functions
- Render callbacks

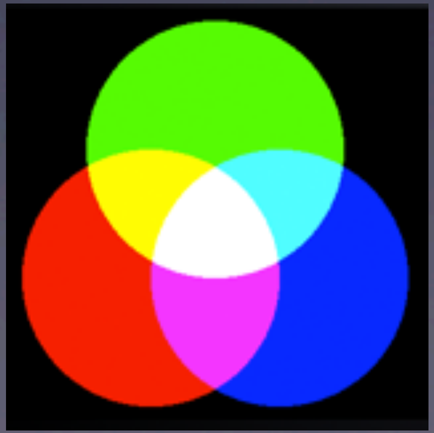


Layer Blending

- Use any OpenGL blending functions
- Example: alpha blend (transparency)
    - (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)

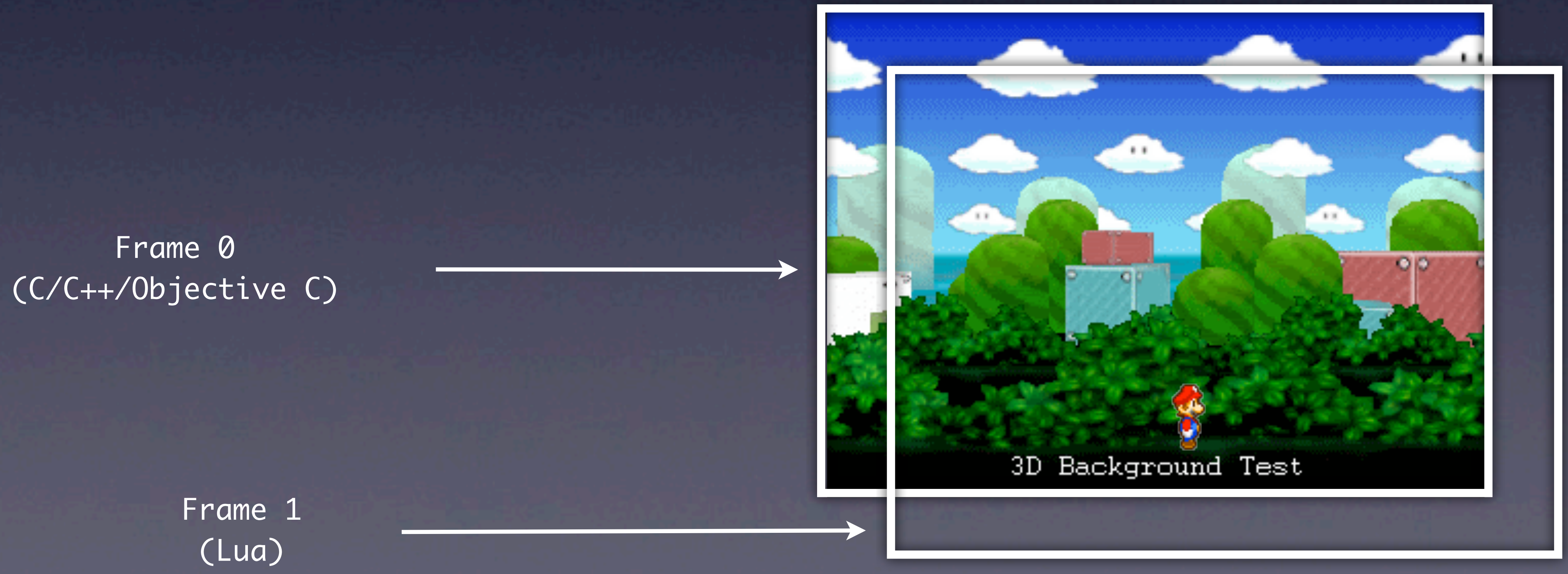


- Example: additive blend (particle system)
    - (GL_ONE, GL_ONE)



Render Callbacks

- Render layers using custom Objective C code / OpenGL
- Example: scrolling background



Events

Events

- EnterFrame - fires every render loop

Events

- EnterFrame - fires every render loop
- Touch - touch events with phases

Events

- EnterFrame - fires every render loop
- Touch - touch events with phases
- Scene events

Events

- EnterFrame - fires every render loop
- Touch - touch events with phases
- Scene events
- Timer - execute code after a delay

Events

- EnterFrame - fires every render loop
- Touch - touch events with phases
- Scene events
- Timer - execute code after a delay
- Physics events - collisions

Graphics Elements

- Sprites
- Lines
- Rectangles
- Circles
- Polygons
- Text via Text Candy™
- Display Groups

Physics

- Uses Box2D Physics
- All graphics elements can have physics properties
- Bodies, fixtures, joints, forces, etc.
- All Box2D body types supported (dynamic, static, kinematic)
- Collision events

Sound

- Based on the ObjectiveAL API
- Sound Effects

- Music

Transitions

- Alter a display objects properties over time
- Position, size, color, etc.
- Can be used to product “canned” animations or effects

Transitions

- Alter a display objects properties over time
- Position, size, color, etc.
- Can be used to product "canned" animations or effects

Scene Management
The Director API

Scene Management
The Director API

- Scenes are collections of layers and the objects they contain

Scene Management
The Director API

- Scenes are collections of layers and the objects they contain
- Scene transition effects
    - slide
    - page curl

Scene Management
The Director API

- Scenes are collections of layers and the objects they contain
- Scene transition effects
  - slide
  - page curl
- Scene management Events
  - createScene
  - *sceneWillEnter*
  - sceneEntered
  - *sceneWillExit*
  - sceneExited
  - destroyScene

Implementation

Implementation

- Libraries of C methods that delegate to methods on Objective C objects

Implementation

- Libraries of C methods that delegate to methods on Objective C objects
- Libraries includes factory method to create objects

Implementation

- Libraries of C methods that delegate to methods on Objective C objects
- Libraries includes factory method to create objects
- Use metatables to define custom "types"
  e.g., `createMetatable(L, GEMINI_RECTANGLE_LUA_KEY, rectangle_m);`

Custom Lua Types

Custom Lua Types

- Wrap Objective C functionality in Lua “objects”

Custom Lua Types

- Wrap Objective C functionality in Lua "objects"
- Create, manipulate, and destroy Objective C objects from Lua

Custom Lua Types

- Wrap Objective C functionality in Lua "objects"
- Create, manipulate, and destroy Objective C objects from Lua
- Custom types defined in C "libraries" with library factory methods as well as instance methods

Representing Objective C Types in Lua

- Userdata - provides a block of raw memory with no predefined Lua operations
  - Can store anything here - we will store pointer to our Objective C object

Representing Objective C Types in Lua

- Userdata - provides a block of raw memory with no predefined Lua operations
  - Can store anything here - we will store pointer to our Objective C object

```
lua_newuserdata(lua_State *, size_t size)
```

Metatables

Metatables

- Tables that can change the behavior of other tables or userdata

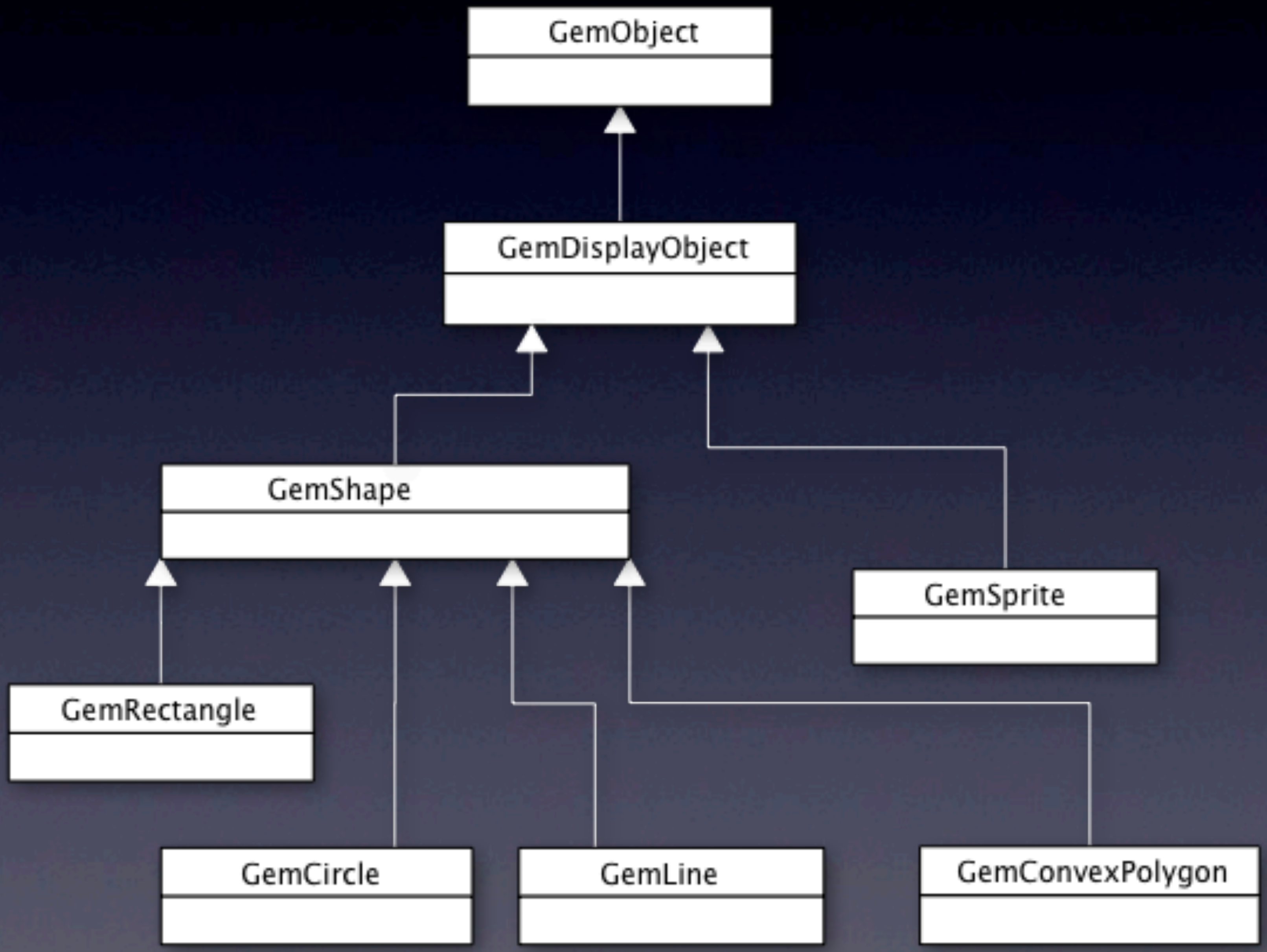
Metatables

- Tables that can change the behavior of other tables or userdata
- Used to assign a ‘type’ to userdata

Metatables

- Tables that can change the behavior of other tables or userdata
- Used to assign a ‘type’ to userdata
- Can hold method mappings for userdata

Gemini Classes



GemObject.h

```
@interface GemObject : NSObject {
    NSMutableDictionary *eventHandlers;
    lua_State *L;
    int selfRef;
    int propertyTableRef;
    int eventListenerTableRef;
    NSString *name;
}

@property (nonatomic) int selfRef;
@property (nonatomic) int propertyTableRef;
@property (nonatomic) int eventListenerTableRef;
@property (readonly) lua_State *L;
@property (nonatomic, retain) NSString *name;

-(id) initWithLuaState:(lua_State *)luaState LuaKey:(const char *)luaKey;
-(BOOL)getBooleanForKey:(const char*) key withDefault:(BOOL)dflt;
-(double)getDoubleForKey:(const char*) key withDefault:(double)dflt;
-(int)getIntForKey:(const char*) key withDefault:(int)dflt;
-(NSString *)getStringForKey:(const char*) key withDefault:(NSString *)dflt;
-(void)setBOOL:(BOOL)val forKey:(const char*) key;
-(void)setDouble:(double)val forKey:(const char*) key;
-(void)setInt:(int)val forKey:(const char*) key;
-(void)setString:(NSString *)val forKey:(const char*) key;
-(BOOL)handleEvent:(GemEvent *)event;
@end
```

GemObject Initializer

```
-(id) initWithLuaState:(lua_State *)luaState LuaKey:(const char *)luaKey {
    self = [super init];
    if (self) {
        L = luaState;

        __unsafe_unretained GemObject **lgo = (__unsafe_unretained GemObject **)lua_newuserdata(L, sizeof(self));
        *lgo = self;



















    }

    return self;
}
```

GemObject Initializer

```
-(id) initWithLuaState:(lua_State *)luaState LuaKey:(const char *)luaKey {
    self = [super init];
    if (self) {
        L = luaState;

        __unsafe_unretained GemObject **lgo = (__unsafe_unretained GemObject **)lua_newuserdata(L, sizeof(self));
        *lgo = self;

        luaL_getmetatable(L, luaKey);
        lua_setmetatable(L, -2);















    }

    return self;
}
```

GemObject Initializer

```
-(id) initWithLuaState:(lua_State *)luaState LuaKey:(const char *)luaKey {
    self = [super init];
    if (self) {
        L = luaState;

        __unsafe_unretained GemObject **lgo = (__unsafe_unretained GemObject **)lua_newuserdata(L, sizeof(self));
        *lgo = self;

        luaL_getmetatable(L, luaKey);
        lua_setmetatable(L, -2);

        // append a lua table to this user data to allow the user to store values in it
        lua_newtable(L);
        lua_pushvalue(L, -1); // make a copy of the table becaue the next line pops the top value
        // store a reference to this table so our object methods can access it
        propertyTableRef = luaL_ref(L, LUA_REGISTRYINDEX);

        // set the table as the user value for the Lua object
        lua_setuservalue(L, -2);








    }

    return self;
}
```

GemObject Initializer

```
-(id) initWithLuaState:(lua_State *)luaState LuaKey:(const char *)luaKey {
    self = [super init];
    if (self) {
        L = luaState;

        __unsafe_unretained GemObject **lgo = (__unsafe_unretained GemObject **)lua_newuserdata(L, sizeof(self));
        *lgo = self;

        luaL_getmetatable(L, luaKey);
        lua_setmetatable(L, -2);

        // append a lua table to this user data to allow the user to store values in it
        lua_newtable(L);
        lua_pushvalue(L, -1); // make a copy of the table becaue the next line pops the top value
        // store a reference to this table so our object methods can access it
        propertyTableRef = luaL_ref(L, LUA_REGISTRYINDEX);

        // set the table as the user value for the Lua object
        lua_setuservalue(L, -2);

        // create a table for the event listeners
        lua_newtable(L);
        eventListenerTableRef = luaL_ref(L, LUA_REGISTRYINDEX);




    }

    return self;
}
```

GemObject Initializer

```
-(id) initWithLuaState:(lua_State *)luaState LuaKey:(const char *)luaKey {
    self = [super init];
    if (self) {
        L = luaState;
        
        __unsafe_unretained GemObject **lgo = (__unsafe_unretained GemObject **)lua_newuserdata(L, sizeof(self));
        *lgo = self;
        
        luaL_getmetatable(L, luaKey);
        lua_setmetatable(L, -2);
        
        // append a lua table to this user data to allow the user to store values in it
        lua_newtable(L);
        lua_pushvalue(L, -1); // make a copy of the table becaue the next line pops the top value
        // store a reference to this table so our object methods can access it
        propertyTableRef = luaL_ref(L, LUA_REGISTRYINDEX);
        
        // set the table as the user value for the Lua object
        lua_setuservalue(L, -2);
        
        // create a table for the event listeners
        lua_newtable(L);
        eventListenerTableRef = luaL_ref(L, LUA_REGISTRYINDEX);
        
        lua_pushvalue(L, -1); // make another copy of the userdata since the next line will pop it off
        selfRef = luaL_ref(L, LUA_REGISTRYINDEX);
        
    }
    
    return self;
}
```

The Library Bindings

The Library Bindings

- Libraries consist of two classes of functions

The Library Bindings

- Libraries consist of two classes of functions
  - Library functions

The Library Bindings

- Libraries consist of two classes of functions
  - Library functions
    - Factory methods

The Library Bindings

- Libraries consist of two classes of functions
  - Library functions
    - Factory methods
    - Methods that effect global state

The Library Bindings

- Libraries consist of two classes of functions
  - Library functions
    - Factory methods
    - Methods that effect global state
  - Object methods attached via metatables

The Library Bindings

- Libraries consist of two classes of functions
  - Library functions
    - Factory methods
    - Methods that effect global state
  - Object methods attached via metatables
    - :setFillColor, :insert, etc.

The Library Bindings

- Libraries consist of two classes of functions
  - Library functions
    - Factory methods
    - Methods that effect global state
  - Object methods attached via metatables
    - :setFillColor, :insert, etc.
    - index, newIndex, __gc, etc.

Important typedefs for Registering a Library

```
typedef int (*lua_CFunction) (lua_State *L);


typedef struct luaL_Reg {
  const char *name;
  lua_CFunction func;
} luaL_Reg;
```

Library Binding Function

```
int luaopen_display_lib (lua_State *L){









}
```

Library Binding Function

```
int luaopen_display_lib (lua_State *L){
    // create meta tables for our various types /////////

   // .... other types not shown for brevity ///////////

    // lines
    createMetatable(L, GEMINI_LINE_LUA_KEY, line_m);

    // rectangles
    createMetatable(L, GEMINI_RECTANGLE_LUA_KEY, rectangle_m);

    /////// finished with metatables ///////////





}
```

Library Binding Function

```
int luaopen_display_lib (lua_State *L){
    // create meta tables for our various types /////////

   // .... other types not shown for brevity ///////////

    // lines
    createMetatable(L, GEMINI_LINE_LUA_KEY, line_m);

    // rectangles
    createMetatable(L, GEMINI_RECTANGLE_LUA_KEY, rectangle_m);

    /////// finished with metatables ///////////

    // create the table for this library and populate it with
   // our functions
    luaL_newlib(L, displayLib_f);

    return 1;
}
```

Display Library Function Mapping

```
// the mappings for the library functions
static const struct luaL_Reg displayLib_f [] = {
    {"newLayer", newLayer},
    {"newGroup", newDisplayGroup},
    {"newLine", newLine},
    {"newRect", newRectangle},
    {"newCircle", newCircle},
    {"newShape", newShape},
    {NULL, NULL}
};
```

Mapping for the Rectangle Methods

```
// mappings for the rectangle methods
static const struct luaL_Reg rectangle_m [] = {
    {"__gc", genericGC},
    {"__index", rectangleIndex},
    {"__newindex", rectangleNewIndex},
    {"setFillColor", rectangleSetFillColor},
    {"setGradient", rectangleSetGradient},
    {"setStrokeColor", rectangleSetStrokeColor},
    {"setStrokeWidth", rectangleSetStrokeWidth},
    {"delete", genericDelete},
    {"addEventListener", addEventListener},
    {"removeEventListener", removeEventListener},
    {"applyForce", applyForce},
    {NULL, NULL}
};
```

Rectangle Factory Method

Lua:

```lua
local rectangle = display.newRect(x, y, width, height)
```

Rectangle Factory Method

Lua:

```
local rectangle = display.newRect(x, y, width, height)
```

C Factory Method:

```
static int newRectangle(lua_State *L){

    GLfloat x = luaL_checknumber(L, 1);
    GLfloat y = luaL_checknumber(L, 2);
    GLfloat width = luaL_checknumber(L, 3);
    GLfloat height = luaL_checknumber(L, 4);

    GemRectangle *rect = [[GemRectangle alloc] initWithLuaState:L
         X:x Y:y Width:width Height:height];
    [[((GemGLKViewController *)([Gemini shared].viewController))
               .director getDefaultScene] addObject:rect];

    return 1;
}
```

Adding Object Bindings

Adding Object Bindings

- Cannot use dynamic libraries on iOS

Adding Object Bindings

- Cannot use dynamic libraries on iOS
- Must use static linking

Adding Object Bindings

- Cannot use dynamic libraries on iOS
- Must use static linking
- Best way to do this is to modify linit.c

Opening Our Library in linit.c

```c
static const luaL_Reg loadedlibs[] = {
  {"_G", luaopen_base},
  {LUA_LOADLIBNAME, luaopen_package},
  {LUA_COLIBNAME, luaopen_coroutine},
  {LUA_TABLIBNAME, luaopen_table},
  {LUA_IOLIBNAME, luaopen_io},
  {LUA_OSLIBNAME, luaopen_os},
  {LUA_STRLIBNAME, luaopen_string},
  {LUA_BITLIBNAME, luaopen_bit32},
  {LUA_MATHLIBNAME, luaopen_math},
  {LUA_DBLIBNAME, luaopen_debug},

  {NULL, NULL}
};
```

Opening Our Library in linit.c

```
extern int luaopen_display_lib (lua_State *L);

static const luaL_Reg loadedlibs[] = {
  {"_G", luaopen_base},
  {LUA_LOADLIBNAME, luaopen_package},
  {LUA_COLIBNAME, luaopen_coroutine},
  {LUA_TABLIBNAME, luaopen_table},
  {LUA_IOLIBNAME, luaopen_io},
  {LUA_OSLIBNAME, luaopen_os},
  {LUA_STRLIBNAME, luaopen_string},
  {LUA_BITLIBNAME, luaopen_bit32},
  {LUA_MATHLIBNAME, luaopen_math},
  {LUA_DBLIBNAME, luaopen_debug},

  {NULL, NULL}
};
```

Opening Our Library in linit.c

```
extern int luaopen_display_lib (lua_State *L);

static const luaL_Reg loadedlibs[] = {
  {"_G", luaopen_base},
  {LUA_LOADLIBNAME, luaopen_package},
  {LUA_COLIBNAME, luaopen_coroutine},
  {LUA_TABLIBNAME, luaopen_table},
  {LUA_IOLIBNAME, luaopen_io},
  {LUA_OSLIBNAME, luaopen_os},
  {LUA_STRLIBNAME, luaopen_string},
  {LUA_BITLIBNAME, luaopen_bit32},
  {LUA_MATHLIBNAME, luaopen_math},
  {LUA_DBLIBNAME, luaopen_debug},
  "display", luaopen_my_math_lib,
  {NULL, NULL}
};
```

Demos

What’s Next?

What’s Next?

- New features

What's Next?

- New features
  - More scene transitions

What's Next?

- New features
  - More scene transitions
  - Particle system

What’s Next?

- New features
  - More scene transitions
  - Particle system
  - Built in text support

What’s Next?

- New features
  - More scene transitions
  - Particle system
  - Built in text support
- Multithreading

What’s Next?

- New features
  - More scene transitions
  - Particle system
  - Built in text support
- Multithreading
  - Physics

What’s Next?

- New features
  - More scene transitions
  - Particle system
  - Built in text support
- Multithreading
  - Physics
  - Scene loading

What’s Next?

- New features
    - More scene transitions
    - Particle system
    - Built in text support
- Multithreading
    - Physics
    - Scene loading
- Support for more third party tools (level builders, etc.)

Questions?

- https://ithub.com:indiejames/GeminiSDK.git
- Blog http://blog.stokedsoftware.com
- Twitter @jnorton
- Box2D - http://box2d.org
- ObjectAL - http://kstenerud.github.com/ObjectAL-for-iPhone/