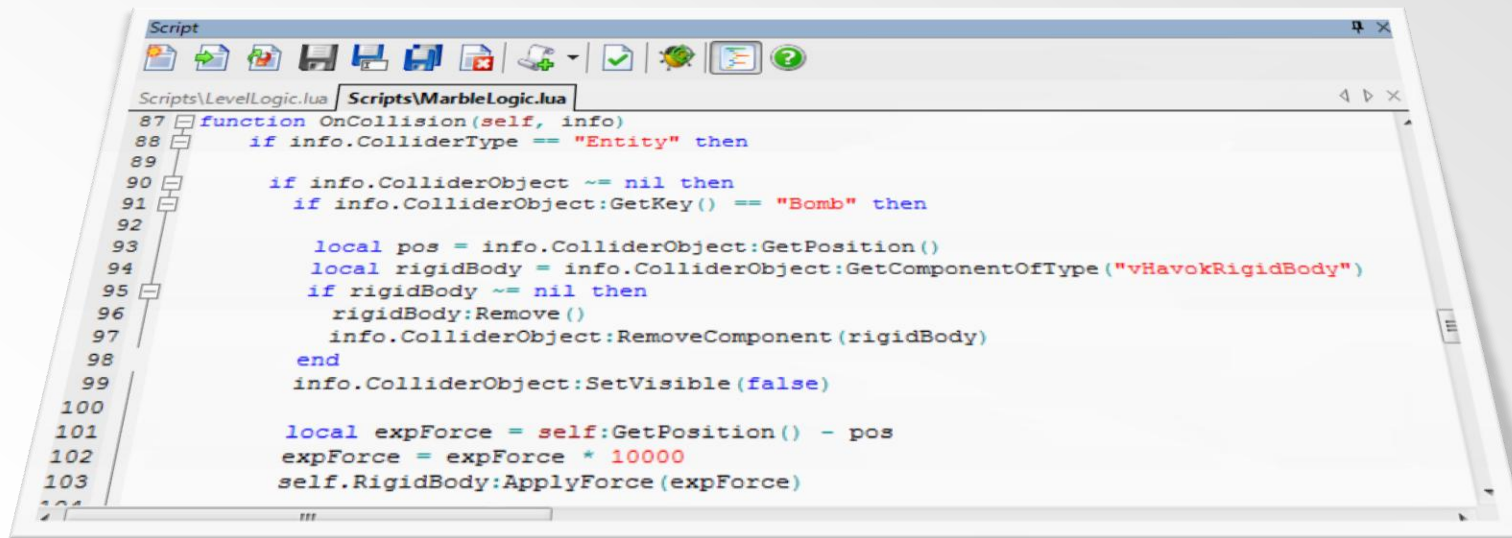
Thomas Pollak : Havok Korea

November 2011, Version 1.1

Lua Prototyping and Module Extensions for the Havok Vision Engine



Lua Prototyping: Agenda

- Overview of the Lua API in Vision
  - Take a look at a simple, fully Lua driven prototype
  - Extending the prototype
- Creating custom extension modules with SWIG
  - Integrating the module
  - Making use of the custom module
- Summary & Outlook

intel

Overview: Lua API

- You can use a Lua script in Vision for…
  - The whole game - GameScript
    Present from initialization to de-initialization of the engine.
  - The current scene - SceneScript
    From scene initialization to de-initialization.
  - GUI actions
    Using callbacks like OnActive, OnClick, OnDoubleClick, …
  - Any scene object by attaching the script component
    Also using callbacks: OnCreate, OnSerialize, OnThink, …

intel

Overview: Lua API

- Working with callbacks I

```
fuction OnAfterSceneLoaded(self)
  self:AttachComponentOfType("vHavokRigidBody", "RigidBody")
  self.RigidBody:ApplyForce(...)
  -- setup a camera
  self.Camera = Game:GetCamera()
  self.Camera:SetPosition(0,-600,900)
  local lookAt = Game:GetEntity("CamProxy")
  self.Camera:LookAt( lookAt:GetPosition() )
  self.Camera:AttachToEntity(lookAt)
end
```

Wrapper object of the parent object. This could be a BaseEntity, LightSource, ParticleEffect, … (of Lua UserData type)

Attach an arbitrary component by name…

Behaves like a table

Access globals like 'Game' to create or search for entities, cameras, particle effects, …

Overview: Lua API

- Working with callbacks II

```
fuction OnCollision(self, collisionInfo)

  if collisionInfo.ColliderType == "Entity" then
    if collisionInfo.ColliderObject:GetKey() == "Bomb" then

      local colliderPos = info.ColliderObject:GetPosition()

      -- apply an explosion force
      local explosionForce = self:GetPosition() - colliderPos
      explosionForce = explosionForce * 10000

      self.RigidBody:ApplyForce(explosionForce)

      Debug:PrintLine("Hit a bomb at: " .. collisionInfo.HitPoint)

    else ...
  end
end
```

A table containing information about the collision, eg: HitPoint, HitNormal, ColliderType, ColliderObject, …

Access the ‘Debug’ global for simple output to the viewport or log, or even debug rendering…

Lua API Module Overview

Sound | Physics | PhysX

Vision:
- Game
- Input
- Application
- Screen
- Timer
- Renderer
- Util
- Debug
- GUI

Overview:Selected Features

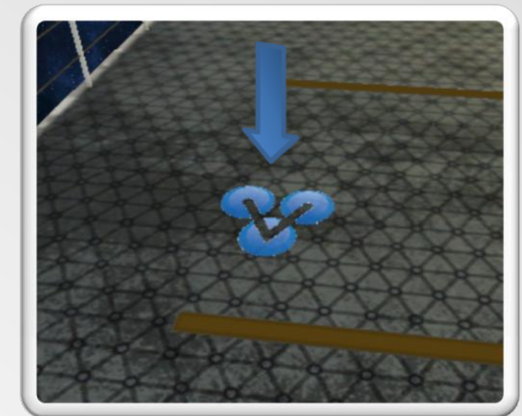
- Using particle effects:

```
local effect = Game:CreateEffect(pos, "Particles/Explosion.xml")
effect:SetWindSpeed(Vision.VisVector_cl(100,0,0), false)
effect:SetMeshEmitterEntity(self)
effect:SetApplyTimeOfDayLight(true)
effect:SetColor(Vision.V_RGBA_RED)
effect:Restart()
effect:GetRemainingLifeTime()
effect:SetIntensity(0.7)
...
```

Overview:Selected Features

- Using shaders and wallmarks:

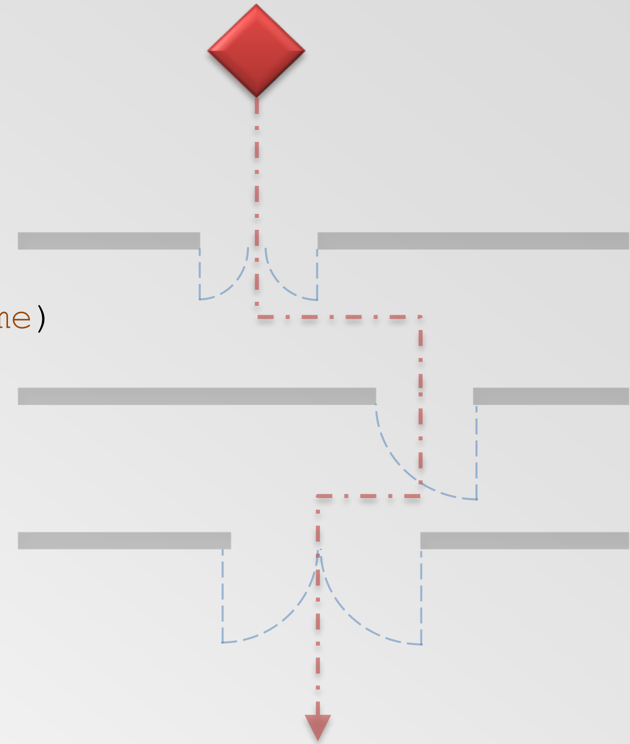
```
self:SetEffect("surface_shield", "DemoShaderLib", "MagicGlow")

-- apply to all present instance via the mesh
local mesh = self:GetMesh()
mesh:SetEffect("surface_sword", "DemoShaderLib", "MagicGlow", "Exp=4")

self:ClearShaderSet()

-- use the renderer instance to draw wallmarks
Renderer.Draw:Wallmark(pos, dir,
        "vision.dds", Vision.BLEND_ALPHA, 100)

Debug.Draw:BoundingBox(self)

Debug.Draw:BoneBoundingBox(self, "Hips")

Debug:Enable(false)

...
```



Overview:Selected Features

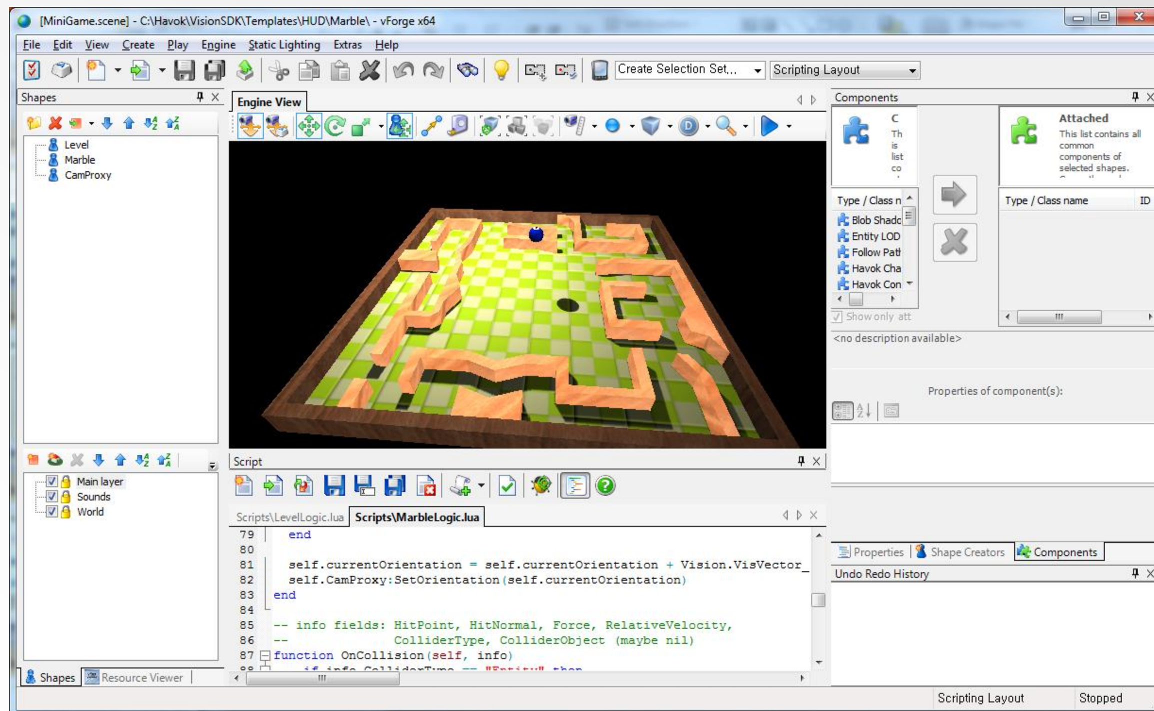
- Using triggers:

```
-- attach this script to gates and doors
function OnCreate(self)
  self:AddTriggerSource("Gate")
end

function OnTrigger(self, sourceName, targetName)
  self.Animation:Play("OpenGate")
end

-- add a trigger target to a main character
function OnAfterSceneLoaded(self)
  self:AddTriggerTarget("Hero")
  self.Hero:LinkToSource("Gate")
end

function OnThink(self)
  if DistanceToClosestSource(self)<80 then
    self.Hero:TriggerAllTargets()
  end
end
```



Lua Prototype: vForge

- Take a look at the prototype
- Extending the prototype
  (with some of the described features)



Creating an Extension Module

- Since Vision 8.2 we are using SWIG (Simplified Wrapper and Interface Generator) to generate Lua wrappers
- SWIG requires interface files to generate wrappers
  (e.g. an interface file for VisBaseEntity to gat a VisBaseEntity wrapper)
- These interface files are always bundled together to a module – remember the Vision, Sound & Physics module?
- Therefore you need also a module for custom wrappers of your own components and entities…

intel

Creating an Extension Module

Vision specific customization

MyLuaModule.i

```
%include <windows.i>
%include <vision_types.i>

%module MyLuaModule

%include < VisApiVector.i>
%include
< VisTypedEngin...
```

→ **SWIG** → **iSwig** → **MyLua Module _wrapper .cpp**

↓

VisTypedEngine Object_cl.i

```
%nodefaultctor VisTyp...
%nodefaultdtor VisTyp...

class TypedEngineObje...
```

VisApiVector.i

```
class VisVector_cl
{
public:
  VisVector_cl ()
  VisVector_cl (float...
```

. . .

Add the file to your Visual Studio Project and call the initialization method `luaopen_MyLuaModule(L)`

intel

Module: Interface Files

- Interface files have basically the same syntax like C++ headers (since you can also use them) with the possibility to add directives starting with ‘%’. Generally it is a good idea to expose just the required functions and methods – no code bloat, easy to understand, …
- Important directives:
  - %nodefaultctor, %nodefaultdtor:
    Avoid the creation of default constructors and destructors for objects managed by Vision (like VisBaseEntity, ObjectComponent, VisTypedEngineObject, …)
  - %extend:
    Allows you to add convenience methods to your wrapper
  - %rename:
    Rename types or methods to give them a short, more convenient name for scripting
  - %native:
    For native LUA C API implmented functions, which are not possible in C++ (returning multiple parameters, mixed return types, …)

Module: Interface Files

- Making an Interface File (HUD element for the current prototype)

```
class HUDScore : public VDialog
{
  public:
    HUDScore(int iInitialLives = -1);
    ~HUDScore();

    void AddScore(int iScore);
    void ReduceLives();
    void Reset();
    bool IsAlive();

    VOVERRIDE void OnPaint(...);

  protected:
    int m_iLives, m_iInitialLives, m_iScore;

    VisFontPtr m_spFont;
};

HUDScore * ShowHUDScore(int iInitLives = -1);
```

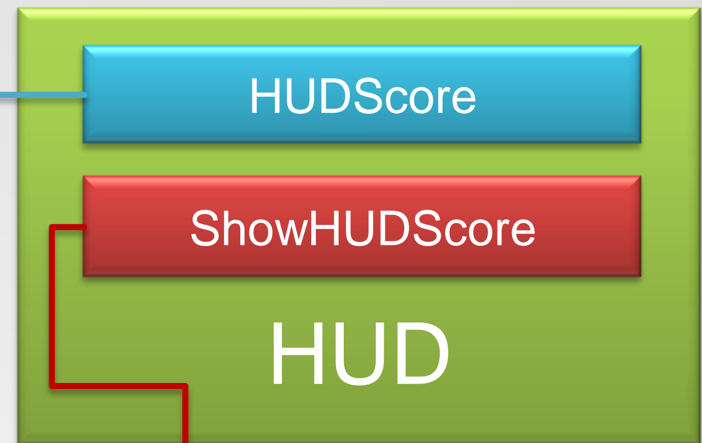
Desired functionality in the Lua wrapper

Module: Interface Files

- Making an Interface File (HUD element for the current prototype)

```
%nodefaultctor HUDScore;
%nodefaultdtor HUDScore;

class HUDScore : public VDialog
{
  public:
    void AddScore(int iScore);
    void ReduceLives();
    void Reset();
    bool IsAlive();
};
```

```
HUDScore * ShowHUDScore(int iInitLives = -1);
```




```
%module HUD
%{
   #include "HUDScore.hpp"
%}

%include "HUDScore.i"
```

intel

Generating the Module

- Running SWIG and iSWIG

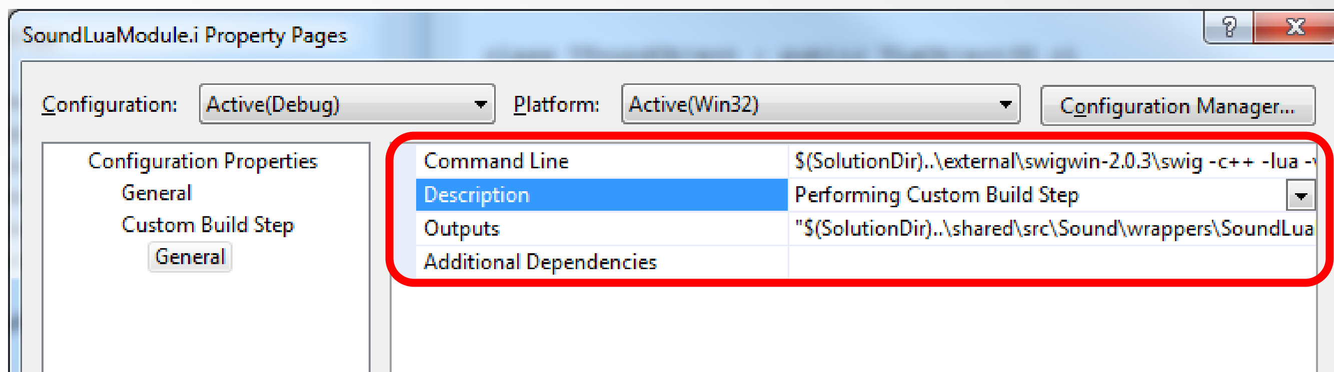
```
swig -c++ -lua -verbose -o MyLuaModule_wrapper.cpp MyLuaModule.i
```

  - Take a look at the output of SWIG, the warnings are usually very helpful, run iSwig afterwards:

```
iswig --include StdAfx.h MyLuaModule_wrapper.cpp
```

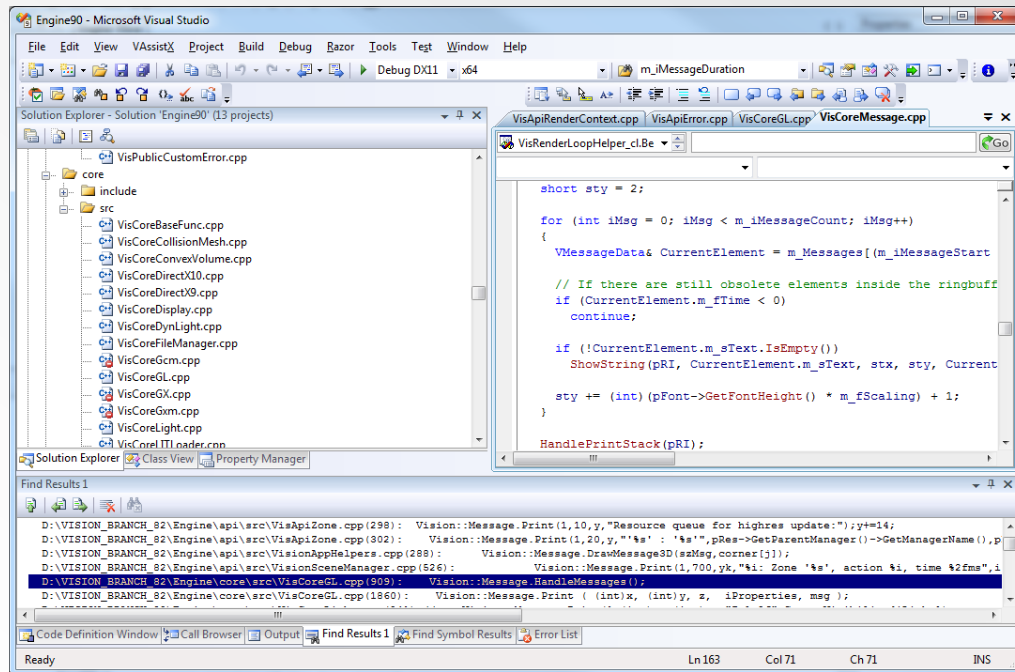
  - Include StdAfx.h in order to use precompiled headers (you can include multiple headers, seperating them with a ';' semicolon)

- Let Visual Studio do the job for you



Module Extension: VisualSudio

- Integrating the module
- Making use of it…
  (HUD element instead of the debug output)



Summary

- Script Types: GameScript, SceneScript, GUIScript, ObjectScripts
- Using callback like OnCreate, OnAfterSceneLoaded, OnDestroy to communicate with the scripts
- The Vision Lua API is bundled in Modules:
  Vision, Sound, Physics and PhysX
- Ready made functionality to work with Entities, Particle Effects, Shaders, Wallmarks, Triggers, Cameras, …
- You can create your own module and wrappers without writing Lua wrapper code. All steps can be fully integrated into Visual Studio.

Outlook

- Introducing an OnExpose callback to make script components accessible via parameters
- Easy access to render specific things like the TimeOfDay component
- Adding support for wide characters
- Increase performance using Havok Script

Questions?

thomas.pollak Φ havok.com