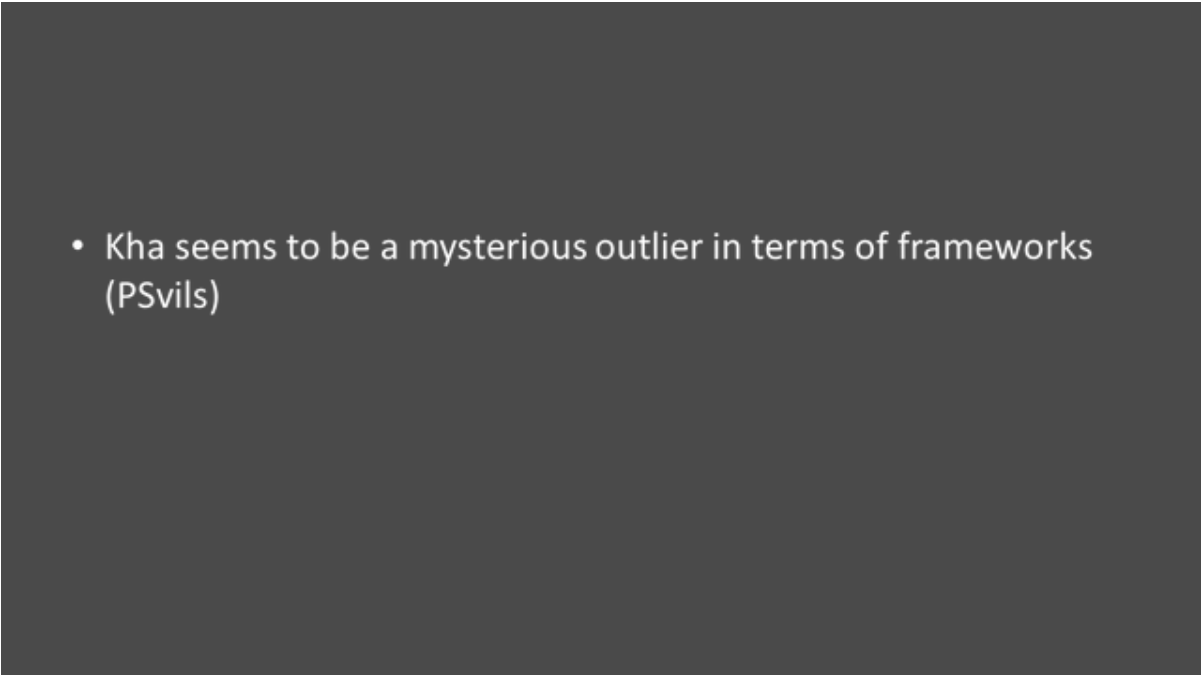


These are the slides from the Kha presentation at WWX2015.  
The comments are new but mostly similar to the talk (the actual talk will be on YouTube later this month).



Kha

It doesn't need any subtitles because it's Kha.

- 
- Kha seems to be a mysterious outlier in terms of frameworks (PSvils)
-

- it's like the shadow assassin guild of Haxe frameworks (Simn)

- `frameworks.sort([f1, f2] => f2.popularity - f1.popularity);`
- OpenFL
- SnowKit
- Kha

The least popular...

- `frameworks.sort([f1, f2] => f2.coolness - f1.coolness);`
- Kha

- Kha
- SnowKit
- OpenFL

but clearly the coolest of all the Haxe game frameworks.

## Rob

- Robert Konrad / @robdangerous
- KTX Software Development
  - Go and buy our games on Steam
    - Leona's Tricky Adventures
    - Redux: Dark Matters
    - The Haunted: Hell's Reach
- Lots of C/C++ and hardly any Flash

Hi, I'm Rob, mind buying our games?

Leona: <http://store.steampowered.com/app/305880>

Redux: <http://store.steampowered.com/app/336930>

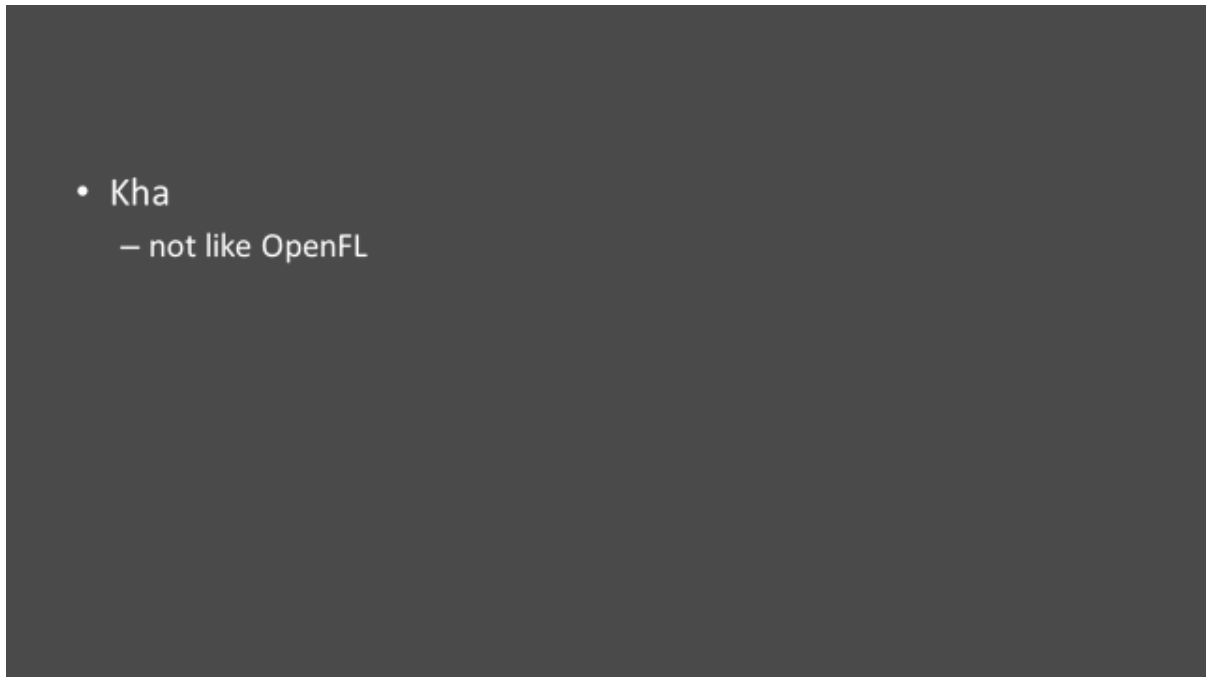
Haunted: <http://store.steampowered.com/app/43190>

So, I'm a C++ kinda guy and therefore...

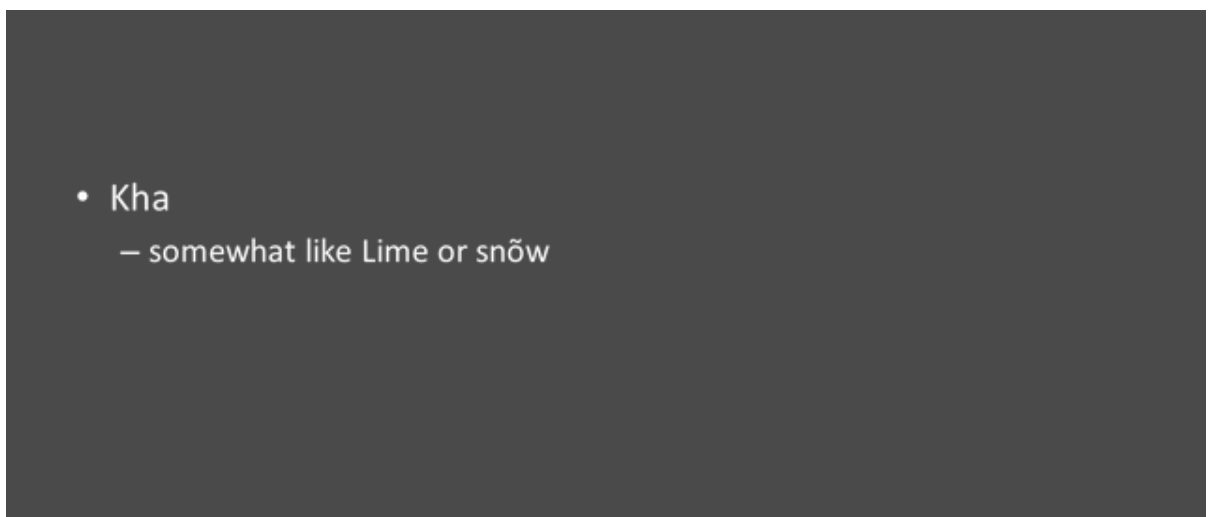
- Kha
  - not like Flash



Kha is not like Flash. Consequently...



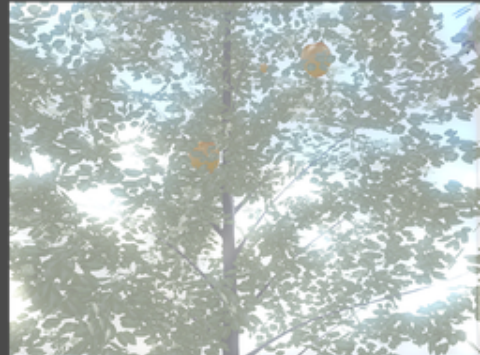
Kha is also not like OpenFL.



It's a low level portability layer, somewhat like Lime or snow.

## Kt

- C++ game engine
- Fancy 3D and stuff
- Very portable
  - Also running on PS360



Kha has two ancestors, first one is Kt, a fancy and portable C++ game engine.


## Portability in C++

- C++ is portable by default
- There's lots of traps
  - `#include <foo\bar.h>`
- And many shortcuts
  - For example D3DX, GLU,...

Speaking of C++, it is pretty portable by itself, when you do it right and avoid using unportable utility libraries which try to hide how stuff works.

## Kje

- Tiny but portable Java game engine
- Designed for education

A screenshot of a 2D platformer game built with Kje. The game features a brown squirrel-like character on a brown platform. In the background, there are green trees and a blue sky with white clouds. Two floating platforms with three red spikes each are visible. The ground is made of brown dirt and stones. At the bottom of the screen, there is a status bar with a highscore of 0, a health indicator with three red hearts, and a timer.

Kha's second ancestor is Kje, a tiny 2D Java game engine, build for educational purposes.

## Kje

- Includes
  - Hardware abstraction api
  - Super basic 2D game engine
  - Super basic 2D tilemap editor
- Task
  - Hack it and make your own game

This was also designed in a portable way...

## Kje portability

- JVM
- Android
- HTML5
  - Via GWT

and worked in the JVM, on Android and in the browser using Google Web Toolkit.

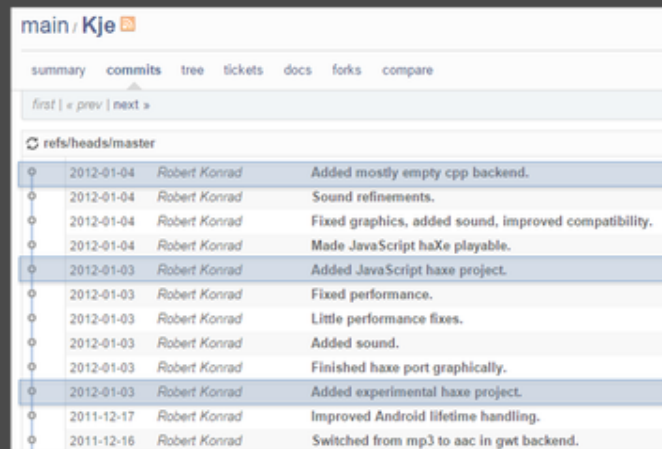
## GWT

- Everything about it is slow
  - The compiler
  - The generated JavaScript

GWT though wasn't that great.



## Kje

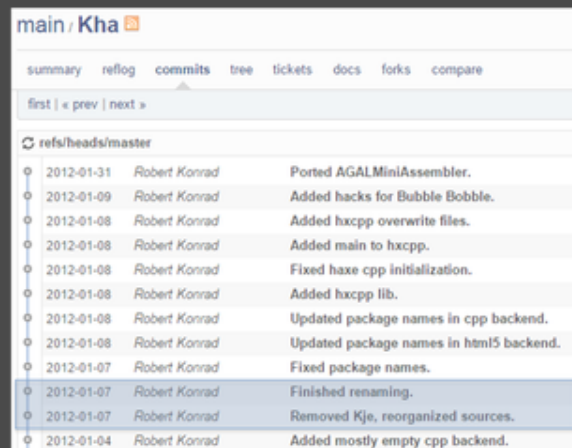


The screenshot shows the commit history for the 'main' branch of the 'Kje' repository. The interface includes tabs for 'summary', 'commits', 'tree', 'tickets', 'docs', 'forks', and 'compare'. Below the tabs, there are navigation links: 'first', 'prev', and 'next'. The commit history is displayed as a table with columns for date, author, and commit message. The commits are listed in reverse chronological order, starting from 2012-01-04 at the top and ending with 2011-12-16 at the bottom. The commit messages describe various changes, including adding a cpp backend, refining sound, fixing graphics, and adding JavaScript and experimental haxe projects.

date	author	commit message
2012-01-04	Robert Konrad	Added mostly empty cpp backend.
2012-01-04	Robert Konrad	Sound refinements.
2012-01-04	Robert Konrad	Fixed graphics, added sound, improved compatibility.
2012-01-04	Robert Konrad	Made JavaScript haxe playable.
2012-01-03	Robert Konrad	Added JavaScript haxe project.
2012-01-03	Robert Konrad	Fixed performance.
2012-01-03	Robert Konrad	Little performance fixes.
2012-01-03	Robert Konrad	Added sound.
2012-01-03	Robert Konrad	Finished haxe port graphically.
2012-01-03	Robert Konrad	Added experimental haxe project.
2011-12-17	Robert Konrad	Improved Android lifetime handling.
2011-12-16	Robert Konrad	Switched from mp3 to aac in gwt backend.

So I was on the lookout for an alternative and did some experiments.

## Kha



The screenshot shows the commit history for the 'main' branch of the 'Kha' repository. The interface includes tabs for 'summary', 'relog', 'commits', 'tree', 'tickets', 'docs', 'forks', and 'compare'. Below the tabs, there are navigation links: 'first', 'prev', and 'next'. The commit history is displayed as a table with columns for date, author, and commit message. The commits are listed in reverse chronological order, starting from 2012-01-31 at the top and ending with 2012-01-04 at the bottom. The commit messages describe various changes, including porting AGALMiniAssembler, adding hacks for Bubble Bobble, adding hxcpp overwrite files, adding main to hxcpp, fixing haxe cpp initialization, adding hxcpp lib, updating package names in cpp and html5 backends, fixing package names, finishing renaming, removing Kje, reorganizing sources, and adding a mostly empty cpp backend.

date	author	commit message
2012-01-31	Robert Konrad	Ported AGALMiniAssembler.
2012-01-09	Robert Konrad	Added hacks for Bubble Bobble.
2012-01-08	Robert Konrad	Added hxcpp overwrite files.
2012-01-08	Robert Konrad	Added main to hxcpp.
2012-01-08	Robert Konrad	Fixed haxe cpp initialization.
2012-01-08	Robert Konrad	Added hxcpp lib.
2012-01-08	Robert Konrad	Updated package names in cpp backend.
2012-01-08	Robert Konrad	Updated package names in html5 backend.
2012-01-07	Robert Konrad	Fixed package names.
2012-01-07	Robert Konrad	Finished renaming.
2012-01-07	Robert Konrad	Removed Kje, reorganized sources.
2012-01-04	Robert Konrad	Added mostly empty cpp backend.

Porting it to Haxe took a day, worked much better for HTML5 and allowed me to merge in parts of Kt to add native backends. Thus Kha was born.

## Kha



- Once just a little experiment
- Now the world's most portable game development platform
- But it's not a game engine
  - Moved those parts out

And look where that ended up. It's now the most portable thingamagic in the world. All higher level game engine stuff was moved out though (more about that later).

## Kha

Portability

Efficiency

Robustness

Education

Kha tries to get four big topics just right, let's have a closer look at each.

# Portability

## Kha

- Uses Haxe to make CPU code portable
- Cross-compiles Haxe code to
  - JavaScript
  - C++
  - C#
  - Java
  - Flash
  - ... (other targets not used by Kha)

Haxe makes cpu-code more portable than anything else.

## Kha

- Uses kfix to make GPU code portable
- Cross-compiles GLSL code to
  - Different flavors of HLSL
  - Different flavors of GLSL

- AGAL
- Metal

krafix makes gpu-code more portable than anything else. krafix is a GLSL cross-compiler.

## krafix

- Successor to kfx (modified version of the ANGLE compiler)
- Based on the GLSL reference compiler and SPIR-V
- Already supports 3 target languages
  - Well prepared for more

krafix is the successor to Kha's previous shader cross-compiler kfx. It is based on SPIR-V and the GLSL reference compiler.

## SPIR-V

- Bytecode for GPU code by Khronos
- Introduced for Vulkan
- krafix compiles GLSL to SPIR-V
- then compiles SPIR-V to whatever

SPIR-V is Khronos' new shader bytecode that was recently announced together with Vulkan (aka GLnext). khrfix cross-compilation capabilities are only loosely coupled to the GLSL reference compiler because they work solely on SPIR-V – therefore easy to support more than just GLSL in the future.

## Current Status

- Brand new
- Lots of missing instructions
- Can already compile very complex shaders
- Tessellation Shaders almost working
- Compute Shaders are next

It's a very new thing with some problems remaining. But it's progressing fast and already implementing up to date features that kfx/angle couldn't provide. Tessellation shader support is almost done, first compute shaders are also working.

## Compute Shaders

- Ignore the rendering pipeline

- Can speed up many calculations
  - Shared memory, atomics,...

Compute shaders by the way make it possible to program GPUs in more general ways easier and also provide more synchronization options to make things faster compared to previous shader types.

## Compute Shaders

- Windows (with included drivers)
  - HLSL
- OSX
  - OpenCL
- iOS
  - Metal
- Android
  - GLSL

But almost every system has its own ideas about what a computer shader should look like. Kha is about to solve that problem...

## Current Status Elsewhere

- Unity
  - No cross-compilation of Tessellation and Compute

- Unreal
  - Everything but OpenCL

Meanwhile Unity is...really, I don't know what they're doing. Unreal does pretty well though, only missing support for OpenCL.

## GLSL in Kha

- Portable dialect of GLSL

GLSL in Kha is a slightly modified version of GLSL to make it more portable. Here are two examples (which currently are all the differences, but there might be more in the future).

## Videos in GLSL

```
uniform sampler tex;  
varying vec2 texCoord;
```

```

varying vec2 texCoord;

void main() {
    gl_FragColor = texture2D(tex, texCoord);
}

```

Hardware accelerated video decoding on most systems doesn't really concern GLSL, because it just returns a pixel buffer that can be copied to a texture.

## Videos in GLSL for Android

```

#extension GL_OES_EGL_image_external : require
uniform samplerExternalOES tex;
varying vec2 texCoord;

void main() {
    gl_FragColor = texture2D(tex, texCoord);
}

```

But on Android hardware video decode writes to a GLOESEGL\_image thingy. Using that looks somewhat silly in GLSL.

## Videos in portable GLSL?

```

#ifdef ANDROID
#extension GL_OES_EGL_image_external : require

```

```

uniform samplerExternalOES tex;
#else
uniform sampler tex;
#endif
varying vec2 texCoord;

void main() {
    gl_FragColor = texture2D(tex, texCoord);
}

```

So a GLSL video shader for all systems looks like this. Not pretty.

## Videos in Kha

```

uniform samplerVideo tex;
varying vec2 texCoord;

void main() {
    gl_FragColor = texture2D(tex, texCoord);
}

```

In Kha accessing a video-texture looks like this.

## Tessellation Control Shader in GLSL

```

void main() {
    tc_pos[gl_InvocationID] = v_pos[gl_InvocationID];
    if (gl_InvocationID == 0) {
        gl_TessLevelInner[0] = inner;
        gl_TessLevelOuter[0] = outer;
    }
}

```



```

    gl_TessLevelOuter[1] = outer;
    gl_TessLevelOuter[2] = outer;
}
}

```

Tessellation control shaders in GLSL contain the code to define per-patch data and per-vertex data inside one function. Usually per-patch calculations are therefore inside one big if block.

## Tessellation Control Shader in Kha

```

void patch() {
    gl_TessLevelInner[0] = inner;
    gl_TessLevelOuter[0] = outer;
    gl_TessLevelOuter[1] = outer;
    gl_TessLevelOuter[2] = outer;
}

void main() {
    tc_pos[gl_InvocationID] = v_pos[gl_InvocationID];
}

```

D3D11 instead uses two functions and so does Kha because that can easily be mapped to D3D11 and GLSL.

## hxsl

- That works, too
- Compiles to GLSL, then calls krafix

hxsl, the Haxe Shading Language by Nicolas Cannasse which allows you to write shaders in Haxe syntax directly inside your regular Haxe code, is also supported (but doesn't yet support all of hxsl's features).

## Lime/snow approach

- No Graphics api
  - Just forwards OpenGL/WebGL
- Sound api
  - OpenAL for native targets
  - Custom implementation for HTML5
- Input api
  - SDL for native targets
  - Custom implementation for HTML5
- Everything else
  - SDL for native targets
  - Custom implementation for HTML5

Now let's look at the apis and their implementations. Lime and snow implement each api twice, once for SDL/OpenAL and once for HTML5. And graphics – they don't really do graphics, they just forward OpenGL.

## Kha approach

- Graphics api
  - Custom implementation for every target
- Sound api

- Sound api
  - Custom implementation for every target
- Input api
  - Custom implementation for every target
- Everything else
  - Custom implementation for every target

Kha on the other hand implements everything for every target. And it has its own graphics api.

## Kore

- Used for all native targets
- The cleaned up „Kore“ of our C++ engine
- Very similar api
- Can also be used separately
  - Purely C++

For native targets Kha forwards calls to a C++ library that looks and works very similar to Kha and can also work on its own. It's called Kore.

Kha (all Haxe)



krafix



The complete structure looks something like this.



Now, more api details. Kha does in fact have more than one graphics api. I call it a generational api design, there is a separate api optimized for different hardware/software generations. Currently it's up to Graphics4 with Graphics5 in the pipeline (Metal is already running as a G4 backend right now though and D3D12 is already half-done, too).



- Minimizes Draw-Calls
- Add-on shader api
- Combined usage of g2 und g4 possible

Additionally there are generic implementations of lower level apis targeting the higher levels. So for every of Kha's target the backend implements just the highest possible api and the generic implementations then make the lower level apis available, too. G2on4 might be especially interesting as it provides a high-performance 2D implementation for current graphics hardware. Also G2on4 provides additional api calls, making it possible to use custom shaders in a 2D scene.

## Graphics2

- Similar to <canvas>, Java's Painter,...

```
g.drawImage(image, x, y);  
g.rotate(angle, cx, cy);  
g.color = Color.Red;  
g.fillRect(10, 10, 100, 100);
```

So, G2 looks like this.

## Graphics4

- Similar to modern OpenGL
- Much easier to work with
  - No legacy stuff
  - One way to do one thing
  - Minimized global state
  - One draw call (drawIndexedVertices)

## Graphics4

```
g.setVertexBuffer(vertexBuffer);  
g.setIndexBuffer(indexBuffer);  
g.setProgram(program);  
g.setTexture(textureLocation, image);  
g.setMatrix(projectionLocation, projectionMatrix);  
g.setBlendingMode(sourceBlend, destinationBlend);  
g.drawIndexedVertices();
```

And G4 looks like this.

g

```
override function render(frame: Framebuffer) {  
    var g = frame.g4;  
    g.begin();
```

```
...  
g.end();  
}
```

The g object seen in the samples can be obtained from the Framebuffer object...

## Render to Texture

```
var image = Image.createRenderTarget(512, 512);  
var g = image.g4;  
g.begin();  
...  
g.end();
```

or from an image. So rendering to a texture and then using that is really easy.

## Audio

- Generational api
- Audio1
  - Simple „play that file“ api
- Audio2
  - Direct access to the audio buffer

- And of course Audio1on2

Audio is now also a generational api.

## Audio1

```
var channel = Audio.playSound(sound);  
channel.stop();
```

Audio1 looks like this.

## Audio2

```
var t = 0.0;  
Audio.audioCallback = function (samples: Int, buffer: Buffer) {  
    for (i1 in 0...samples) {  
        buffer.data.set(buffer.writeLocation, Math.sin(t));  
        buffer.writeLocation += 1;  
        t += 0.001;  
    }  
});
```



---

And Audio2 looks like this. Using Audio2 you can basically do anything you want.

## Audio2on1

- Software mixer
- `haxe_stb_ogg_sound`  
by shohei909

Audio1on2 is a generic software audio mixer and includes ogg playback support (therefore Kha can support ogg playback in for example Safari).

## Input

- Straight forward callback based api
- Keyboard, touch, mouse, gamepad

Input...

## Input

```
Gamepad.get(0).notify(axisListener, buttonListener);  
function axisListener(axis: Int, value: Float) {  
    switch (axis) {  
    case 0:  
        if (value < -0.2) left = true;  
    }  
}  
...
```

just callbacks.

## Scheduler

- Handles timing
- Schedule events per frame or in absolute game time
- Scheduler.addTimeTask(function() { ... }, 0.1);

The Scheduler is a global object that handles all timing. Callbacks can be scheduled to be executed at a specific game time or per frame.

---

## project.kha

- Sets global options for the game
- Optionally lists all assets
  - With compression options
- Assets can be assigned to rooms for easier loading

Assets are handled using a per project config file called project.kha.

## project.kha

```
"assets": [
  {
    "type": "image",
    "file": "grass.png",
    "name": "grass"
  },
  {
    "type": "image",
    "file": "pirate.png",
    "name": "pirate"
  }
],
"rooms": [
  {
    "name": "all",
    "parent": null,
    "neighbours": [],
    "assets": [
      "grass",
      "pirate"
    ]
  }
]
```

Here is an [actual example](#).

## Loader

- Easy asset loading
  - `Loader.the.loadRoom(„all“, init);`
- Manual asset loading
  - `Loader.the.loadImage({name: „pirate“, file: „pirate.png“}, arr);`

In the actual game assets can optionally be loaded in packages called rooms as defined in `project.kha`.

## khamake

- Converts/optimizes assets
  - included tool for images
  - configurable tools for audio and video
- Creates FlashDevelop projects and `hxml` files
- Creates additional IDE projects where it makes sense
  - directly for non-native targets
  - by calling `koremake` for native targets

`khamake` is Kha's build tool which reads in `project.kha` and optimizes/compresses all assets for the chosen target platform. It also compiles all Haxe and GLSL code and creates `hxproj` and `hxml` files.

## koremake

- Creates IDE projects
  - Visual Studio, Xcode, Code::Blocks, ...
- Similar to cmake and premake

koremake is the accompanying build tool for Kore. It's a generic C++ build system but doesn't directly build the project but instead outputs IDE projects. That's one more step to compile the project, but so much more possibilities – debuggers, profilers,...

## Really more portable than anything else?

- SDL
- Ported to more systems
- C instead of C++ and manual memory management
  - Good for old systems
- No cross-compilation
- No 3D graphics
  - No real contender

So how portable is it really, compared to other portable stuff? SDL disqualifies because it doesn't handle graphics.

## Really more portable than anything else?

- Monkey
- Comes close
- Less competent in graphics

Monkey is nice but not really up to date when it comes to graphics.

## Really more portable than anything else?

- Eqla?
- Outracks Uno?
- Don't know

And those things? I really have no idea.

## Some Target Details

## HTML5

- g4 on WebGL
- g2 on canvas
- a2 on WebAudio
- a1 on audio element
- + HTML5-Worker target

The HTML5 target is really really good. Full G4 support, well performing fallback to canvas,...

Also there is a second HTML5 target. The HTML5-Worker target runs the complete project inside a worker thread and outputs rendering commands to the main thread. Great for avoiding timeouts in long calculations. Also great for embedding Kha inside for example a web-based game editor.

Kore/native

- g4 on every target
- a2 on every target

The Kore backend is very straight forward – g4 and a2 on all native targets.

## Windows

- g4 on D3D9, D3D11 and OpenGL
  - D3D12 half done
- Does not need d3dx or d3dcompiler at runtime
- Win 10 Universal apps already supported

Some native targets actually support multiple graphics backends. Windows supports three (and soon four). Also, full Win10 support and no d3dx libs needed at runtime (so no directx installation for your clients).

## iOS



- OpenGL or Metal
- pvrtec textures

iOS target supports OpenGL and Metal. Also pvrtec texture compression (just add compressed: true to an image in project.kha).

## Android

- Native target + Java target
  - But Java target currently outdated
- Compressed textures in progress...

There are actually two Android targets, although the Java target is currently not up to date. Support for all the different kinds of compressed textures on Android currently in progress.

## Java

- g2 on Painter
- a1 on javax.sound

On the other end of the spectrum are things like the Java backend. This only supports g2 and a1, but works solely on standard Java apis and can therefore be embedded in Java applications without hassles.

## Consoles

- Xbox 360 and PlayStation 3 backends do exist
- A little outdated though
- If interested, get in touch

Old backends for Xbox 360 and PlayStation 3 do exist, but currently no devkits at hand. If interested to use that and get it up to date, please get in touch.

## Efficiency

## Efficiency

- Expose low level apis as directly as possible
- Generally does as little as possible

Kha is efficient because it doesn't do much. All apis are designed to map to the underlying hardware/software as directly as possible

## Efficiency

- `Loader/project.kha` makes it easy to

- load just the necessary assets
- keep assets organized
- Use texture compression

Also khamake handling assets automatically can help a lot with loading times and memory usage.

## Efficiency

- khamake/koremake's tendency to create IDE projects makes it easy to use a plethora of performance tools

Plus because khamake/koremake create IDE projects you have all native tools at your disposal to inspect performance further.

## Bunnies



Have [some bunnies](#)

Sources at <https://github.com/KTXSoftware/BunnyMark>

Please note the simplified implementation compared to the original version: <https://github.com/KTXSoftware/BunnyMark/blob/master/Sources/TileTest.hx>

It's just calling drawImage for every bunny. Kha can make that fast automatically in most situations.

## Current Status

- Robust performance
- Easy to optimize for specific use cases
- Always room for improvements
  - Also in Haxe and krafix

Performance is robust but it gets really fast when you optimize for your usecase. G2onG4 makes that easy for 2D – take it and change it. Rotating bunnies can be made much faster by calculating the rotations inside the vertex shader for example

(but it's not a good optimization for all situations, for that reason it's not in Kha's G2onG4).

## Robustness

## Robustness

- Favors git based distribution using submodules
  - Every Kha project versions every dependency
    - Even the Haxe compiler
- Slightly modified Haxe compiler
  - No hidden downloads
- Aim:  
Check out project from 10 years ago,  
create the same binary bit for bit

Kha projects are usually handled using git submodules so that every dependency is properly versioned. Even the Haxe compiler itself is just a submodule.

---

## Robustness

- khamake/koremake's tendency to create IDE projects makes it easy to use all kinds of debuggers,...

## Depends on C++ compiler

- „Check out project from 10 years ago, create the same binary bit for bit“
- Sadly can't include C++ compilers and sys libs

C++ compilers and stuff – just too big and often proprietary.

## Education

## Education

## Education

- Kha always contains all code
  - No precompiled libs at all
  - Additional compile time negligible
    - Because Kore is small
- khamake/koremake create IDE projects by default
- debugging always works
- „goto definition“ always works

Introspection, all the way down, always working.

## Education

- Google'n'Paste programming hardly possible
- Underlying concepts clearly represented in the api
- Api is redesigned when it makes sense
  - See deprecated package
  - See „Breaking News“



Other code can't just be pasted in, because Kha doesn't clone anybody's apis. So you have to understand what you are doing. But Kha makes that easy, because the apis represent the underlying concepts very clearly.

## Education

- Kha is small and tidy
- Easy, layered design

:)

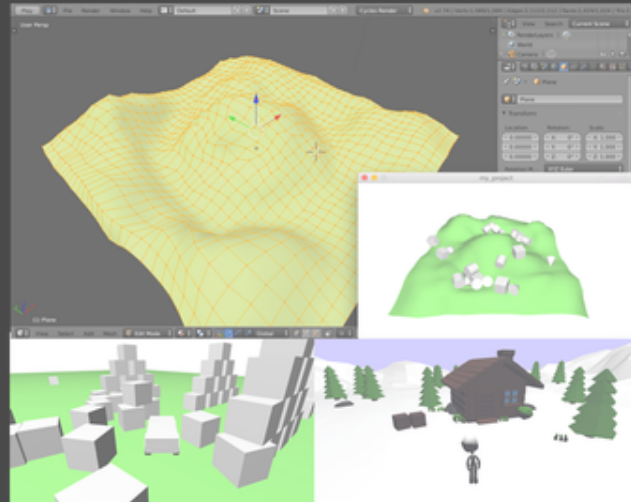
## But

- Documentation is still lacking

:(

## ZBlend

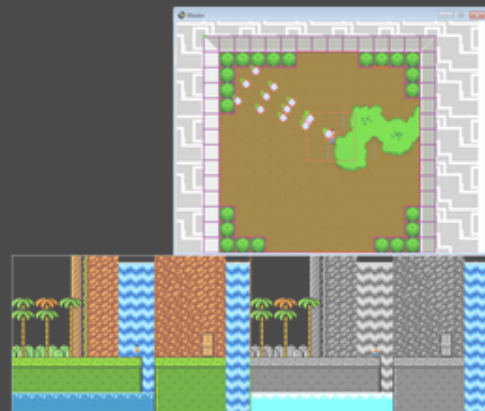
- 3D game engine by Lubos Lenco
- Bullet physics
- Blender as game editor
- <http://zblend.org/>



Now some examples of stuff that runs on top of Kha. ZBlend is a 3D game engine by Lubos Lenco. It includes the bullet physics engine and it integrates with Blender. Even game logic can be defined directly in Blender. <http://zblend.org>

## KhaPunk

- Port of HaxePunk by Sidar Talei
- Supports custom shaders, Tiled levels, nape physics
- <https://bitbucket.org/stalei/khapunk>



KhaPunkt is a 2D game engine – a port of HaxePunk, done by Sidar Talei. Has some nice additions compared to the regular HaxePunk. <https://bitbucket.org/stalei/khapunk>

## Kha3D/KhaUI

- Not yet usable

Those exist on GitHub. Please don't use them. Work in progress.

## Kha2D

- 2D game engine
- Formerly part of Kha
- Took it out to promote „competition“

But this one you can use. It was formerly included in Kha, but I

took it out, because I think everybody should choose for themselves which engine to use on top of Kha – if any.

## Kha resources

- [kha.technology](http://kha.technology/) / [kha.ktxsoftware.com](http://kha.ktxsoftware.com)
- Blog, Forum, (Wiki)
- IRC channel #kha at ktxsoftware.com

All the links are at <http://tech.ktxsoftware.com> – also join our IRC channel (and I'm also usually on #haxe at freenode).

## Kha.VR

- By Florian Mehm
- Extension for building VR applications with Kha
- Currently: Different VR SDKs
  - E.g. time warp-based rendering vs. Direct rendering
- Provide a uniform interface to heterogeneous VR devices

And now for the WXX surprise features: Virtual Reality support

by Florian Mehm.

## Current State

- 4 Backends
  - Samsung Gear VR
  - Oculus Rift
  - Google Cardboard
  - „Emulated“ → for development and testing on PC
- Example Content
  - SteamQuest

Supports basically all devices that are currently available.

## Samsung Gear VR

- Released in late 2014
  - Uses Oculus Technology
- 
- Rendering based on Asynchronous Timewarp
    - <https://www.oculus.com/blog/asynchronous-timewarp/>
    - Render the current view to a texture for each eye and submit to SDK

## Oculus Rift DK 2

- Released in 2014

- Released in 2014
- Consumer Version announced for 2016

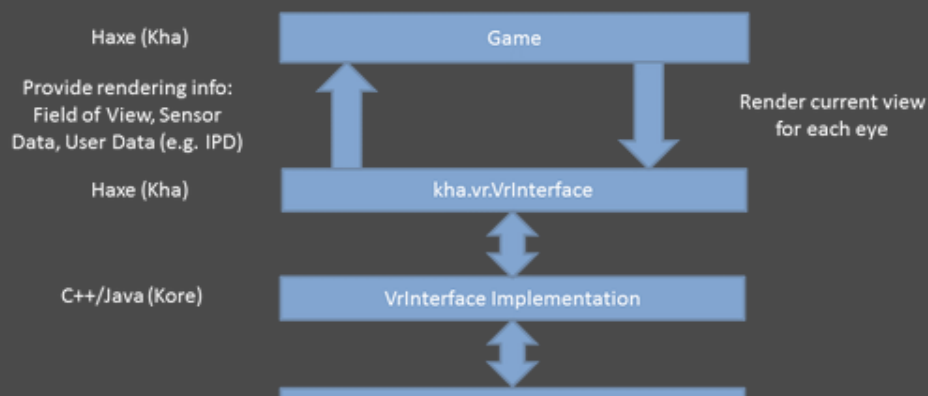


## Google Cardboard

- Low-cost alternative, good introduction to VR
- Released in 2014
- Newer version announced this week



## Architecture



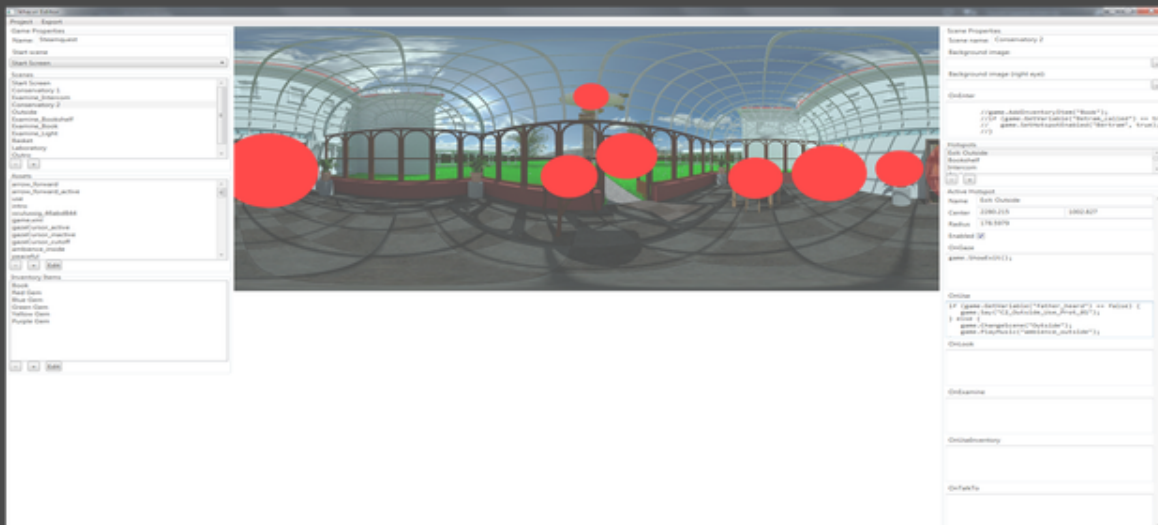
## SteamQuest

- First-Person Adventure Game Demo (think „Myst“)
- → Comfortable experience for most users, good VR introduction experience (no movement, only looking around)



Watch the [SteamQuest trailer](#).

## SteamQuest Engine



## SteamQuest Engine

- Renders 360° stereoscopic images
- Hotspots for interaction (changing the screen, using, lookin, ...)
- Game logic scripted using Haxe
  - Based on hscript <https://code.google.com/p/hscript/>
- Manages project.kha

## Current State and Future Work

- Cleaning up for release/merging with main Kha repository
- VrInterface needs to be more universal
  - Currently based off Gear VR API (Timewarp, Render to Texture, which is not the best way for all SDKs)
- More information will be up soon on [www.mehm.net](http://www.mehm.net)
- Find the current state at <https://github.com/SpookyFM>

The other surprises were [Multiplayer support](#) and [Unity export](#) (there were no slides for those to avoid leaks).