openFrameworks 1.0 Roadmap

by Adam Carlucci, Akira Hayasaka, Arturo Castccro, Atsushi Tadokoro, Chris Sugrue, Christopher Baker, Cyril (Kikko) Diagne, James George, Jordi Puig, Joshua Noble, Kenichi Yo3neda, Kyle McDonald, Lauren McCarthy, Lukasz Karluk, Noriko Matsumoto, Marek Bereza, Roy Macdonald, Satoru Higa, Theo Watson, and Zach Lieberman,

Outline

- Outline
- About this Document
  - Goals of the Roadmap
  - Roadmap Format
- Design
  - Goals and Philosophy
  - Structural
    - Multiwindow and global state
    - Naming conventions
    - Functions, static classes, and OOP
    - Abstraction and Inheritance
- Core
  - 3D
    - Mesh
- Graphics
    - Cairo
    - Scene graph
    - Pixel density independence
    - Internationalization
    - Typography
  - OpenGL
    - Texture
    - Lights and Materials
    - Shaders
  - Sound
  - Platforms and Architectures
    - App Architecture
    - Cross-Platform Experience
    - C++ 11
  - Libraries
  - Video
- Addons
  - 3D Model support
  - OpenCv
  - Network
- Tools
  - Project Generator

- Addon Generator and Manager
- Documentation
  - Examples
  - Reference
  - Tutorials
- Community
  - openframeworks.cc
  - ofxaddons.com
  - Leaders
  - GitHub Contributions
  - Continuous Integration and Testing

About this Document

Goals of the Roadmap

The goal of the openFrameworks developers conference at YCAM is primarily to complete this document, which describes the philosophy and rationale for features of openFrameworks 1.0. This document will be linked to from the website after being reviewed and edited by openFrameworks developers.

The language used in this roadmap must be descriptive (OF should be / should have) and not speculative (OF might do / might be). Any speculative language should be placed in a “Discussion” subtitle.

Roadmap Format

Gray background on text is used to indicate that it is fairly complete and representative.

Light red background on text is used to indicate that something is imminent with the release of 0.9.0.

The headings are used to indicate a hierarchy of topics, and the outline above is autogenerated from those headings. Each topic is broken into multiple chunks that distinguish philosophical descriptions from implementation details, or other information. To clearly communicate the goals of the roadmap it’s important to break things up into the following sections:

Philosophy

Description of the philosophy behind this section.

Implementation

The low level details of how to achieve the goals of this section’s philosophy.

Discussion

Points of the implementation that still have to be discussed.

Milestones

If there is an order to the implementation, it can be mentioned here.

Reference

Any links that help explain the philosophy or implementation details.

Design

Goals and Philosophy

Philosophy

openFrameworks is guided by a number of goals: it should be collaborative, usable and simple, consistent and intuitive, cross-platform, powerful, and extensible. openFrameworks is also driven by a “do it with others” (DIWO) philosophy.

openFrameworks development is **collaborative**. It thrives on the contributions of many people, who engage in frequent discussion, and collaborate on addons and projects. We encourage people to make openFrameworks their own, and contribute to the ecosystem.

openFrameworks tries to balance **usability** and **simplicity**. The earliest versions of openFrameworks used the core as a tool for teaching C++ and OpenGL, but over time the examples have become the best way to learn while the core takes advantage of more advanced features. In exchange, we created many more examples that come with openFrameworks, with the goal of trying to make simple, hackable starting points for experimentation.

openFrameworks is **consistent** and **intuitive**: it should operate on the principle of least surprise, so that what you learn about one part of openFrameworks can be applied to other parts of it. Beginners can use openFrameworks to learn about common programming patterns, and advanced users will be able to apply their experience from other languages and toolkits.

openFrameworks is a **cross-platform** toolkit. openFrameworks supports as many development environments and operating systems as possible. When you download openFrameworks, you can pick your platform and development environment of choice, and have projects and examples ready to learn from and play with. Difficult to port code is kept out of the core, and kept in addons instead.

openFrameworks is **powerful**: it allows you to leverage advanced libraries like OpenCV, use hardware like your graphics card efficiently, and connect peripherals like cameras and other devices.

openFrameworks is **extensible**. When you find something missing from openFrameworks, it's easy to create addons that extend it. The core addons for openFrameworks generally wrap libraries rather than solving problems in novel ways. When openFrameworks wraps libraries, the libraries are left exposed for further hacking.

The driving philosophy behind openFrameworks is ”**do it with others**” (DIWO). We practice DIWO through workshops, developer conferences, hackathons/labs, knitting circles and meetups in person, and online in the form of mailing lists, forum posts, and so on. The most important thing we want to stress is that you are not alone, that there’s a great group of people out there learning, teaching, hacking, making and exploring the creative side of code.

If you are not a part of the group, welcome!

Structural

Multiwindow and global state

Philosophy

Multiwindow will change the architecture of openFrameworks a bit but we should be careful to not break compatibility with old projects as much as possible.

Mainly we'll use more OOP since we'll need to keep around more windows and applications and all of them will have their own states, events... But whenever it's possible we should still provide the current global functions which will refer to the current window being updated/drawn. Another possibility is that those global functions refer to the main window only or to a particular window, but we should be careful that that doesn't introduce too much complexity.

For some platforms (mobile) multiwindow won't make sense so we should take care that the chosen patterns make sense for those platforms too.

Implementation

Multiwindow will allow 3 possibilities:

- shared context among windows
- 1 context per window, all windows running on same thread
- 1 context per window, each window running on it's own thread

There's 3 classes that are involved in this:

- The application (ofApp/testApp)
- The window knows about the hardware, window size, GL context and contains the ofEvents
- The renderer drawing backend

To allow for shared context (since some GL objects can't be shared among contexts) we need 1 renderer per window. The proposed architecture will consist of 1 application related to a window from which it gets events and 1 renderer for each of those windows which provides the drawing backend for that application

Other objects will be able to attach themselves to events as we do now, but then the events will be in each window instead of being something global

Global functions, like ofBackground, ofGetWidth() will affect one of those windows/renderers only. That window/renderer will be the current one.

ofAppRunner will become a class (a singleton) where we register windows and applications before start or on runtime. The proposed model will look like:

```
int main(){
	ofMainApp * mainApp = new mainApp;
	ofAppRunner().addApplication(mainApp, “mainApp”);
	ofGuiApp * guiApp = new guiApp;
	ofAppRunner().addApplication(guiApp, “guiApp”);

	ofAppRunner().start();
}
```

This will create a default window per application but we can explicitly pass a specific window with settings:

```
int main(){
	ofMainApp * mainApp = new mainApp;
	ofAppGLFWWindow * mainWindow = new ofAppGLFWWindow;
	mainWindow->setSamples(16);
	ofAppRunner().addApplication(mainWindow, mainApp, “mainApp”);

	ofGuiApp * guiApp = new guiApp;
	ofAppGLFWWindow * guiWindow = new ofAppGLFWWindow;
	guiWindow->setSamples(0);
	ofAppRunner().addApplication(guiWindow, guiApp, “guiApp”);

	ofAppRunner().start();
}
```

Applications are assigned a name when they are registered in the appRunner so later on, they can be recovered globally by name instead of index which would be confusing.

Each application has access to it’s own window through a getWindow() method and we can access globally any window through it’s application by name.

Although global access to this classes is not a very good practice we provide this so beginners, mostly people coming from processing, still have a way to access the application.

For reasons of backwards compatibility we’ll still provide the global functions in ofAppRunner which will call the corresponding method on the current window.

Global functions in ofGraphics will follow a similar pattern, we keep them for backwards compatibility and they will call the corresponding function in the current renderer.

With mouse and keyboard events, there could be two models: callbacks for the main window, and for the current window.

Naming conventions

Philosophy

We once tried to move OF to namespaces by using each of the folders (app, 3d, gl, etc.) but it was the wrong approach. The right way to use namespaces is to disambiguate between things with similar names. Some examples:

```
of::stringSplit()
of::stringReplace()
of::drawEllipse()
of::background() or of::setBackground()
of::enableAlphaBlending()
of::VideoPlayer
of::Color
```

For 1.0 we need to remove "`using namespace std`" from the OF core and move it to all the example file .cpp

Boolean getters are always prefixed with "is" or "has", never "get". It can often be a sign of an anti-pattern if you are getting information from a class about its internal state and manipulating it based on that information.

The meaning of set/enable/toggle is:

- enable/disable: exactly 2 states. however, there is no reason to use enable/disable instead of set.
- set: 2+ states (for example, setBlendMode is correct, setVerticalSync is correct).
- toggle: some things with 2 states (yes to fullscreen, no to depth testing).

More notes here: https://github.com/openframeworks/openFrameworks/issues/1547

"Starting" and "ending" functions should be used as their base names ("load", not "loadFile"). The meaning of "starting functions" is:

- `setup`: information is completely contained in the input parameters.
- `load`: implies setup with retrieval from an external resource.
- `allocate`: implies setup with allocation of memory.
- `open`: should be removed (in `ofxKinect`)
- `init`: should be setup (e.g., `initGrabber`)
- `connect`: should be `setup` (in `ofArduino`)

The meaning of "ending" functions:

- `exit`: is *only* used for `ofBaseApp`
- `close`: opposite of setup and load
- `clear`: opposite of allocate
- `unload`: should become close (`ofShader` and `ofSoundPlayer`)
- `disconnect`: should be close

Any ending functions should always called by the destructor. You should never have to call `close()` or `clear()` in your `ofApp`'s `exit()` if everything is created in the stack, see RAII

pattern.

Functions, static classes, and OOP

Philosophy

We should have `ofImageFile` and `ofSoundFile` for interfacing between files and images/sounds. This means refactoring `ofSaveImage()` and `ofLoadImage()` into:

```
ofImageFile img(“img.png”);
ofPixels pix;
img.readTo(pix); // read from disk
img >> pix; // alternative
ofPixels pix;
img.writeFrom(pix); // save to disk
img << pix; // alternative
```

And the direct replacements would be:

```
ofImageFile(“file.png”).readTo(pix);
ofImageFile(“file.png”) >> pix; // alt
ofImageFile(“file.png”).writeFrom(pix);
ofImageFile(“file.png”) << pix; // alt
```

- This follows the `ofFile` pattern. This also removes the need for static methods. Static methods as wrappers should be removed from all classes where constructors and methods can be used instead.

OF has a few procedural functions or static methods that are available, but only for manipulating objects that are not native to OF (such as vectors or strings).

```
of::stringSplit(“a,b,c”) // procedural, using string
of::sort(v) // procedural, using vector
of::string(“a,b,c”).split(); // not this
of::vector(v).sort(); // not this
```

Everything else is done by instantiating an object and calling its methods:x

```
ofFilePath::isAbsolute(string); // not this
ofFilePath(string).isAbsolute(); // new style, more consistent with all interfaces
ofFilePath path=ofFilePath(“dir/”) + ofFilePath(“/file”); // new style
```

Sample discussion from the beginner's advocate:
"What is the ::?"
"Oh it has three meanings, in this case that's a static method of the class."
"Um, static is what happens when I rub a balloon on my head."
"Oh, create an object and ask it a question then."

Every argument in OF should be const if possible, and the methods should all be marked const when that is the case. Some situations where const seems impossible to use are actually rectified by using the "mutable" keyword (primarily for caches that don't change the way an object looks to users).

```
class Object {
public:
	// Returned reference value can't be directly modified
	const float& thing(float& x);
	// Object class itself isn't internally modified
	// (member variables marked "mutable" are exempted)
	float& thing(float& x) const;
	// "Object" class member can't modify `x`
	float thing(const float& x);
};
```

Discussion of static class/members vs. namespace + globals
http://stackoverflow.com/questions/1434937/namespace-functions-versus-static-methods-on-a-class

Abstraction and Inheritance

ofBaseHas* classes should not exist, the *has* hints to to aggregation and we are converting

Philosophy

that into inheritance through those base classes.

Implementation

This means removing classes like ofBaseHasPixels: anything that has Pixels should have a getPixels() method that returns Pixels&. Any class that was receivin1.0 roadmapg a ofBaseHasPixels was probably storing it internally in a vector or using something related to its pixels. From now on it will just receive its pixels and store them internally/use them.

eg:

```
vector<ofBaseHasPixels*> v;

void addClassWithPixels(ofBaseHasPixels & hasPixels){
      v.push_back(&hasPixels);
}

void clearAll(){
      v[i].getPixels()[j]=0;
}
```

becomes:

```
vector<ofPixels*> v;

void addClassWithPixels(ofPixels & pixels){
      v.push_back(&pixels);
}

void clearAll(){
      v[i][j]=0;
}
```

We did this at some point because we didn't have classes like ofPixels so it was hard to store references to them because you lost things like their width and height but with wrapper classes like ofPixels the ofBaseHas* classes become unnecessary and make the inheritance hierarchy too complex

Base classes in OF should either have pure virtuals or a sane default method. If it's an "optional" method, it can have an empty implementation (but this should be rare). For someone extending this classes is later easier to know which methods they are missing to implement if they have empty implementations in their own classes instead of having to check one by one the base class

Discussion

Switching some of these things, like removing ofBaseHasPixels, is a hard break in compatibility. In order to avoid a situation where we break a significant number of addons and estrange a large chunk of the community (like Processing 2.0), we need to brainstorm more about clever ways of maintaining some compatibility, or an ability to deprecate things.

Core

3D

Mesh

Philosophy

Mesh provides a way to store collections of 2D and 3D geometry, associated colors, surface normals, texture coordinates and face information. This data can be easily drawn to a screen using

Implementation

Winged-Edge Mesh, allows for selecting and manipulating edges, vertices and faces.

Discussion

Current State: We have a fairly good mesh system for OF, but we are missing some more advanced features. Should we include these in 1.0?

-> this all are 2d primitives which make more sense inside ofPath than as primitives, then they can be treated as primitives by getting the tessellation in an of3dPrimitive. ofPath allows to use this primitives in renderers that support primitives like cairo or nvidia path rendering, openvg... So it's important to keep 2d primitives on ofPath

- ofStripPrimitive, makes a ribbon/mesh-strip from a polyline (provided a thickness and face normal)

- ofCirclePrimitive
- ofRingPrimitive

Other advanced feature suggested during YCAM of dev conference:

- 3D geometry in OF to 3D printer (3D manufacturing support)
- basic mesh manipulation (possibly boolean operations)
- 3dTexture to mesh (and viceversa)
- PCL - Point Cloud Library support as addon.
- ofxAssimpLoader needs updating to latest version and the addon itself needs some love.

typdef ofMesh ofPointCloud? or similar, perhaps contain ofMesh and allow points only. Some people that come from 3d applications (cinema 4d...) are used to using meshes and it's really weird to see a mesh as something without faces

Graphics

Cairo

Current State: Cairo hasn't quite kept up with the core changes. It needs a thorough going through to make sure there is 1:1 parity where possible.

Goal for 1.0:

- support for viewports larger than the window.
- 1:1 parity with GLRenderer where possible

Scenegraph

Being able to have instances of a geometry (of3dPrimitive) so we don’t duplicate resources

three.js as reference, really clear api

Pixel density independence

Philosophy

Pixel density independence deals with the issue of rendering OF content exactly the same across retina and non-retina devices, and across any other pixel scale modes which may emerge in the future.

Currently on iOS devices and new-generation MacBook Pros, the retina feature means that 1 pixel on a non-retina device equals 4 pixels on a retina device under its default setting. Apple deal with different pixel densities by using **'points'** for dimensions instead of pixels, so all calculations across non-retina and retina devices will be exactly the same and so will be the rendered content, irrespective of the different pixel dimensions. OF should implement the same strategy and deal with dimensions in terms of points, but also support the ability to work in pixels.

We should be careful to take into account other platforms apart from apple, Android has a much -more complex ecosystem so it's probably a better reference as long as it doesn't make things more complex than necessary for other platforms in which case it should be the exception. MS Windows High DPI support may also be considered.

Implementation

A branch has been created on the OF github that has been started during the YCAM dev con, https://github.com/openframeworks/openFrameworks/tree/feature-pixel-density

ofAppBaseWindow now has a couple methods for setting / getting pixel density which is initially set by the device (currently on iOS, the pixel density is set to 2.0 when the device goes into retina mode). From there the pixel density value is propagated throughout the OF framework. It can be accessed globally via ofGetPixelDensity() and ofSetPixelDensity().

Currently the GL renderers deals with OpenGL geometry and pixel density really well.
The modelView matrix inside ofMatrixStack scales up everything by the pixel density and everything renders exactly the same across non-retina and retina devices.
ofGetWidth() and ofGetHeight() values have also been fixed to return in points, rather than pixels (iOS only at the moment).

The next challenge is to make pixel density work with ofFbo and ofTexture.
This also applies to fonts, images and anything else that uses textures.
The issue here is that ofFbo's and ofTexture's need to be created at retina sizes. If we are in retina mode on the iPhone and the window resolution in **points** is 320x480, we need to create

these objects at double the resolution, 640x960. But when it comes to rendering these objects, instead of scaling up the geometry of the backing quads, we need to do the reverse and scale down by the pixel density, so they render correctly to size. For this to happen ofTexture and possibly ofFbo needs to carry a pixel density property value which it can use to adjust the quad size when rendering.

The getWidth() and getHeight() methods will need to return point sizes and not pixel sizes. One issue is that these methods are also used internally in ofTexture etc., and internally these classes need to get pixel dimensions and not points. We will have to create some public methods for returning dimensions in point sizes and internally use methods that use pixel dimensions.

ofTrueTypeFont is a special case since it already allows for pixel density when loading it. The pixel density is actually specified in DPI so we should check if we can get the DPI for the screen that the application is running on. This is different to how we are treating the pixel density right now which is a multiple of a reference implementation and is perhaps too specific to one platform (apple / retina)

Discussion

Further discussion on this topic is happening on github,
https://github.com/openframeworks/openFrameworks/issues/1379

Discuss what it's actually pixel density, a multiple of a reference implementation or dpi

Milestones

It is realistic to have this implemented for OF 0.9.1 release.

Internationalization

Philosophy

openFrameworks is an international project and must provide easy-to-use, standards-based international text / string support.

Implementation

- Comprehensive Unicode support is provided by the International Components for Unicode (ICU) library
    - While Poco and the c++ standard library offer some Unicode support, even Poco recommends using ICU : “For reliable multilingual case conversion (and collation, etc.), it's best to use a specialized third-party library, e.g. ICU” (source)
    - Combined, the ICU’s collection of static libraries (including a comprehensive database of Unicode data) is approximately 20-30 MB per platform uncompressed. Linux shared libraries are available via package managers.
- The default encoding for strings will be UTF-8
    - All functions that receive `std::string` (e.g. `ofToUpper()`) will assume UTF-8 encoded text.
    - Utility functions for converting between UTF-8, UTF-16, UTF-32, ISO/IEC 8859-1, etc will be provided by ICU and exposed in the openFramworks StringUtils API.
    - `ofBuffer()`, `ofFile()`, etc will assume UTF-8 content with the option of converting to other encodings upon read / write, etc.
- openFrameworks will support / wrap the most common functions in a new StringUtils namespace that will contain global functions providing a thin wrapper for more complex ICU functions.
- While internationalization is a priority, comprehensive localization mechanisms are not an immediate priority.
    - Users interested in these features should seek platform specific solutions. The development of addons leveraging existing solutions (e.g. gettext) with these capabilities are encouraged. Simple cross-platform solutions are already underway (example).

Discussion

- A goal of openFrameworks is to minimize the number of required 3rd party libraries. While ICU offers a comprehensive suite of very capable tools, it is still not *completely* clear that ICU is the best route to go.
- Initially it seemed that harfbuzz, etc., required ICU, but more recent versions of harfbuzz can be compiled with an internal stripped-down Unicode database sufficient for shaping.
- Thus, it may be sufficient to create an amalgam of `Poco::UTF8::*` functions for Unicode text transformations, digit information, UTF-8 codepoint traversal, etc.

- Ultimately, we have to balance our goal of number of 3rd party libraries in the core with our desire for state-of-the-art text open source text shaping and layout.

Milestones

- Full Unicode integration
- Initial Implementation of UTF-8 compatible StringUtils
- Existing ofUtils string utilities (e.g. `ofToUpper()`) will point to StringUtils impl.

Typography

Philosophy

Typographic layout across languages is complex and very difficult to do correctly. Moving forward, openFrameworks should provide a well-considered set of base classes / interfaces for common typographic tasks with support for rasterization / vectorization of all Unicode codepoints. openFrameworks should provide basic support language internationally sensitive text shaping and layout. The openFrameworks typographic interfaces should provide addon developers with the ability to develop drop-in replacements for the basic implementations provided with openFrameworks in order to leverage complex 3rd party / native tools.

Implementation

The following describes some of the features and technical specifications of the basic typographic implementation and interface. Developers are encouraged to interact with os-native (e.g. OSX) capabilities and 3rd party libraries (e.g. Pango) in addons via the openFrameworks interface system.

- Font Management / Configuration
  - A font management system (e.g. `ofFontManager`) should provide shared access to font resources, rather than loading / rasterizing each time for each class.
- Rasterization / Vectorization
  - OpenType / TrueType rasterization / vectorization is provided via freetype 2.
- Shaping
  - Bidirectional text support via fribidi.
    - This will allow us to deal with non right-to-left languages.
  - Unicode shaping via harfbuzz.
    - This will allow us to shape international characters, deal with ligatures, horizontal and vertical kerning, etc.
- Layout
  - Simple custom implementation in openFrameworks.
    - Correct Line-breaking is relies on Unicode data. Linebreaking algorithms are available in ICU, but are also available in a lightwieght format via liblinebreak.
    - Metrics
      - per-character positioning info
      - block size metrics
    - Simple markup-based layout.
    - Complex kerning via shaping engine / freetype
    - Block layout
      - within `ofRectangle`

            - within `ofPath`
            - Horizontal and Vertical Alignment and Justification
        - Path layout
            - Along `ofPath` / `ofPolyline`
            - Horizontal and Vertical Alignment
- Rendering
    - PDF Rendering in Cairo will continue to use the existing vector outline approach, with the added benefit of the more advanced layout features outlined above. With the improved layout interface, Cairo rendering will also be able to leverage Pango and other more sophisticated layout engines via addons.
    - GL Rendering in openFrameworks will continue to use a texture atlas-based approach, with major improvements. Texture atlases will be managed by the `ofFontManager` (or a implementation of the interface). We are currently using freetype-gl as a reference implementation and will consider the benefits / disadvantage of using the library directly.
        - For latin character sets, most / all glyphs could be pre-loaded into a texture atlas using rectangle packing.
        - For east-asian or other consistently-sized glyphs, a fixed grid-approach would make it more practical to dynamically rasterize and cache glyphs (based on one of these algorithms) and swap glyphs, thus avoiding the problem of re-packing the texture atlas. Rectangle packing algorithms do not offer much of an advantage for consistently-sized glyphs.

Discussion

- The remaining open questions all revolve around the integration of ICU into the core.
    - If ICU is integrated into the core, liblinebreak along with fribidi and harfbuzz (if we are satisfied with ICU's older shaping and bidi implementations) could be avoided.
- freetype-gl is a great reference implementation. It is tempting to simply include this 3rd party library. But it is not feature-complete from our perspective and the library itself already duplicates many of our carefully crafted classes (`ofVbo`, etc). Thus, the best approach seems to be the reimplementation of code with our classes, making some of the features (like bin packing), more easily accessible to developers for other purposes.

Milestones

- The existing `ofTrueTypeFont` should easily support any Unicode codepoint.
- Complete design of the Font Management Interface
- Complete design of the Shaping / Layout Interface
- Complete design of the Rendering Interface
- Implement the basic texture, vector font manager (reference implementation)
- Implement bidirectional shaping w/ fribidi.

- Implement advanced shaping w/ harfbuzz.
- Implement basic block / line layout engine.

References

- **State of Text Rendering**, Behdad Esfahbod

OpenGL

Texture

`ofTexture` is fairly robust at this point, though it is quite a mess in terms of its internals.

Milestones

- enable mipmaps (separate from texture compression)
- support all texture types including loop compressed types.

Lights and Materials

Philosophy

ofLight and ofMaterial should be implemented in shaders for the programmable pipeline. The objects themselves will manipulate different shaders in the programmable pipeline to create lights and materials using the same end-user interface as the current ofLight and ofMaterial objects. Also this will provide an easy way for more advanced users to add their own light shading or material functions into the programmable pipeline without disrupting the rest of the programmable pipeline.

Shaders

Philosophy

Shaders should have clear examples that walk a beginner from creating a simple fragment shader to using fragment+vertex shaders to using uniforms, to finally passing textures and data and doing displacement. A tutorial should accompany a new set of shader examples that will help illuminate the basic concepts of shading and rendering. The shader examples should be in a separate folder containing only shader examples.

Implementation

Note: New shader examples have been added during YCAM OF dev conference, which can be found in tutorials/shader/ folder in OF.

Double up of new tutorial shader examples and existing shader examples in gl/ folder should be removed.

More advanced shader examples still need to be added into gl/ examples folder, some candidate examples are listed below.

- displacement maps with normal calculations for lighting - @julapy todo a simplified version (https://vimeo.com/56965948)
- lambert lighting (http://www.lighthouse3d.com/opengl/glsl/index.php/index.php)
- deferred rendering (https://github.com/jacres/of-DeferredRendering, http://cl.ly/1A0n2j1f3x1g)
- SSAO
- shadow mapping.
- many point lights (http://www.youtube.com/watch?v=crHWEJ-xq3s&feature=youtu.be)
- One example app that can load in any shaders at runtime, based on this great addon => https://github.com/tado/ofxGLSLSandbox

Examples should have explanatory text overlaid on top of the shader with callouts for what's important in the example.

Make a larger model for tutorials in downloads that can be markdown, some code, maybe SVG graphics so as not to bloat the download itself but provide something else in the downloads that can provide some significant step by step instruction that a new user can start working with to understand techniques and approaches.

Examples should leverage as much as possible the new programmable renderer, showing end users how they can leverage the beginCustomShader() functionality to ease their own development.

Discussion

Further discussion happening on github,
https://github.com/openframeworks/openFrameworks/issues/1396#issuecomment-20572970

Milestones

- Tutorial examples have been completed. (DONE)
- Each tutorial needs to be documented and explained. (0.9)
- Copy of written tutorial needs to be included with tutorial examples. (0.9)

Sound

Philosophy

openFrameworks should provide sound functionality which is simple and intuitively mirrors other areas of its functionality. For example, users should be able to get raw samples from an mp3 file as easily as loading pixels from an image file, or play it as easily as drawing an image.

It should offer a pluggable object oriented approach to working with the DSP signal path to allow easily reusable addon code but still provide the low-level functionality of working directly with the buffer.

The sound system should empower users by providing a minimal barrier to entry, then allowing them to easily integrate other systems as their apps grow in complexity.

Implementation

ofSoundBuffer should be the basic primitive which underlies much of the sound system. It should function as a low-overhead wrapper around a block of interleaved floats stored in memory (i.e. as a vector<float>). It should also provide a set of utility functions which are useful for dealing with audio data, such as interpolation, trimming, analysis and conversion utils which ease integration with external systems.

example:

```
myBuffer[7] *= 5; // access like a std::vector<float>
myBuffer.trimSilence();
```

ofSoundPlayer should handle efficient streaming from on-disk files using platform-specific sound utilities. This default ofSoundPlayer should be complemented by a “basic” ofSoundPlayer implementation which handles playback of an entirely in-memory ofSoundBuffer (by just keeping track of an internal playhead and responding to audio render calls). In this way, the default ofSoundPlayer handles the common use-case of “background music” or other simple-but-crucial functionality, and the “basic” ofSoundPlayer affords a greater level of control for instrument-style usage.

FMOD should be removed from ofSoundPlayer and replaced with native functionality.

ofSoundFile should provide an abstraction for loading and saving of audio files. It should be able to produce an ofSoundBuffer as a one-shot load, or by streaming. ofSoundFile should also be able to provide metadata about files, such as retrieval of ID3 tags. It should be able to take

an ofSoundBuffer object and write it to disk as a wave file (.wav).

example:

```
myFile("some-sound.mp3") >> myBuffer;
myBuffer >> ofSoundFile("output-file.wav");
```

ofSoundObject should be the basic DSP primitive, which allows chaining of objects into DSP graphs. It should be a simple abstract class which only requires that implementations be able to provide ofSoundBuffers on demand. Pre-made implementations of ofSoundObjects such as synthesizers, effects and analysis nodes should only be provided in examples, and should not be part of the core itself. ofSoundObject should be the primary integration point for sound addons.

ofSoundMixer should serve as the termination point for ofSoundObject graphs, and should be a simple additive mixer.

example:

```
class myWickedFilter : public ofSoundObject
{/* implement process() and you're done */}

mySoundPlayer.connectTo(myWickedFilter).connectTo(ofGetSystemSo
undMixer());
```

As much as possible, sound objects should try to replicate the API of their graphics or video openFrameworks counterparts. This means that ofSoundBuffer should mimic ofPixels, ofSoundPlayer should mimic ofVideoPlayer and ofSoundFile should mimic ofImage.

Discussion

- What ofSoundObject functionality is crucial for addon writers?
- What use-cases do the two ofSoundPlayer implementations miss?
- What graphics functionality is missing its sound counterpart?

Reference

I've implemented almost all that is mentioned here on [ofxSoundObjects]

Platforms and Architectures

App Architecture

Philosophy

openFrameworks should leverage the latest hardware architectures across all platforms.

Discussion

- On OS X 64bit support means using a much more restrictive video api.
- Need to recompile all libraries OF uses for 64bit
- Need to inform addons developers the drop of support of OSX 32bit
- Apothecary will help this transition.

Milestones

- 64bit only OS X release
- 64bit only Windows release
- Drop 32bit support on Linux / OS X / Windows

Cross-Platform Experience

Philosophy

openFrameworks should provide an API that makes it easy to make minimal changes to application code in order to run on all supported platforms.

Discussion

- We should have cross-platform gestures (swipe, pinch to zoom, and more).
- We should have cross-platform sensors (GPS, Accelerometer, and more).
- We should integrate mouse input and touch input. The best way is to have mouse callbacks, touch callbacks, and have them both call a set of “gesture” (or “pointer”/”cursor” callbacks).
    - from W3C regarding spec:
        - http://www.w3.org/TR/pointerevents/
    - Notes from MS regarding specification for combined input: http://msdn.microsoft.com/en-us/library/ie/hh673557(v=vs.85).aspx

Milestones

- TBA

C++ 11

Philosophy

openFrameworks should leverage the latest software technologies across all platforms.

Discussion

- Reference: https://github.com/openframeworks/openFrameworks/issues/2577
- XCode 3 is unsupported.
- XCode 4.2 with 10.6 is unsupported.
- But it's just not enabled, using the .xcconfig.
- ofPtr should be deprecated & replaced by shared_ptr and unique_ptr (which is c++11 core).
- Using C++11 in the core is ok before 1.0, but not immediately.
- It might make sense to use the move operator in some places.
- On OS X, full c++11 support will eventually mean switching over compilers from GCC to clang, since only clang / libc++ support full c++11 in XCode.

Milestones

- C++11 will be used minimally in the core, but will be enabled in all platform configuration and project files (via default compiler flags, etc).

Libraries

Philosophy

openFrameworks at its core is a collection of outstanding 3rd parties libraries held together by a set of carefully designed bridge classes and functions that orchestrate novel interactions between these libraries and end user code. The addition of 3rd party libraries adds features, but also makes compilation across platform and staying up-to-date quite challenging. Thus, we must strike a balance between the number of 3rd party libraries shipped with the core and the features they offer. In some cases the right solution is to extract portions of other generously licensed open source libraries with proper attribution.

Discussion

- Currently, libraries have to be hand built for the most part. This makes the process of upgrading versions of libraries slow. Also because we share header files between platforms, if a library upgrade is needed for one platform it means all other platforms need to upgrade. This presents challenges when default packages (such as on Debian/Ubuntu) are well-behind the state of the art.

Milestones

- Streamline 3rd party library upgrade process
    - Use / Improve Dan Wilcox's Apothecary system for quick and painless upgrades.
- Consider ways we can reduce OF's dependency on 3rd party libraries.
    - Further leverage existing libraries (e.g. Poco)
    - Host custom up-to-date ppas for Linux
    - Better leverage native shared libraries / features.
- Upgrade all 3rd party libraries to latest versions.

Video

Philosophy

Video playback and capture is a fundamental feature of openFrameworks. What makes openFrameworks unique from other platforms is that it is easy to access the underlying textures and pixels with very few lines of code. We also support flexible use of media, making few assumptions about how the media will be used and offering tools for manipulating position, time, speeds to allow for creative uses and interactivity with video.

openFrameworks supports two major use-cases of video playback: display and processing. Playback allows for asynchronous loading and real time viewing of multiple high definition videos within an openFrameworks application. The processing use case is a frame-to-frame method, allowing an application to access each frame of a video to perform image manipulation and analysis techniques.

OF understands that many of its users need video capture for the purposes of computer vision. This means that control over settings such as manual exposure, color, frame size, and rate are important. OF provides methods to allow for controlling camera settings. OF should also provided methods for saving video camera input to disk as movie files.

There are many different libraries for video capture and playback, each with their own unique advantages and limitations. OpenFrameworks provides a common interface to swappable underlying implementations interfacing with many of these libraries. These means other and future third party libraries will be easily integrated with openFrameworks.

Implementation

- Wrapping of native libraries for capture and playback.
    - OS X:
        - AVFoundation as default library
        - Legacy QTKit and Quicktime as a swappable alternative
        - Option of GStreamer if the user opts for library installation
    - Windows
        - Quicktime as default library for playback and videoInput for capture
        - Option of GStreamer if the user opts for library installation
        - Intent for Windows Media Framework for DX renderer in the future
    - Linux
        - GStreamer as default

- Loading videos, playback, and frame access
    - Should allow for asynchronous mode as to not see noticeable glitches, but also synchronous loading to guarantee video availability immediately after loading
    - When loading videos the developer can specify if they want pixel access, texture access, or both. By default both are available. The underlying implementation will try to optimize based on this configuration.
    - The developer should explicitly set the video to frame by frame mode if they wish to access frames synchronously for processing. (For example when they want a loop that advances one frame each iteration accessing the pixels on each iteration)
    - Support for native video pixel formats, such as YUV, when possible.

- Capture
    - Developer needs manual control over camera settings through a programmatic interface.

Discussion

As we move to the new native frameworks, how do we continue backwards compatibility with the legacy quicktime expectations such as synchronous frame grabbing?

We still don't have a good approach on OS X for doing manual exposure control. UVC, A general interface in ofVideoGrabber?

Milestones

- Refactor players and grabbers to support the processing vs display modes, using a setFrameByFrame method to enable synchronous calls to pixels
- Create a ofVideoFrame object that encapsulates native video frames/textures, which will work more seamlessly with the underlying platform libraries.
- Break players and grabbers into core addons, make sure the default implementation is included by default on each platform
- This will enable us to have an 'ofLite' template allowing 64bit windows and mac
- Create an AVFoundation and GStreamer windows/mac implementation

Addons

Philosophy

The difference between ofAddons and ofxAddons is about whether the structure of the addon needs to be tinkered with and explored, and how common or useful the addon is to the OF community. ofxAddons are more volatile, and ofAddons are less volatile and most helpful. Anything that uses the swappable pattern (sound, video, renderers) should be made into an addon. The project generator includes all the addons that make up the current core by default. Platform specific code will be broken out into addons and added to your project by the project generator. This pattern is already implemented by ofxiOS and ofxAndroid.

Implementation

Discussion

MIlestones

- Bring anything that requires subclassing in
    - Require a rewrite of ofxAssimpModelLoader)

3D Model support

Current State: We are using an outdated version of Assimp. Currently there is not an easy way to get model animation into OF (only Maya and Blender can export animated collada models for Assimp).

Discussion:

- Consider updating Assimp to the latest version
    - (Would require a rewrite of ofxAssimpModelLoader)
- Consider an addon for FBX support.
    - Though a closed format, it is considered much more reliable and stable than collada
    - Watch out : Autodesk EULA are super-restrictive. Blender has an FBX exporter but had to drop the FBX import because of this

Goal for 1.0:

-

OpenCv

Current State:

- We have a fairly solid but old ofxOpenCV

Discussion:

- Consider adding ofxCv or a OpenGL C++ api to ofxOpenCV ?
- Consider what new features we can add to ofxOpenCV / ofxCv ?

Goal for 1.0:

-

Network

Current State

- ofxTcpManager and ofxUDPManager are not as robust / feature rich as they could be.

Goal for 1.0:

- Upgrade ofxNetwork to use Poco::Net
- Add NetUtils
    - ofxICMP aka Poco::ICMPClient
        - Note, raw socket access requires root privileges. Implementing this as a part of the core api might confuse people. It could be implemented w/ a datagram socket, but this is not the best solution. This is probably best left as an ofxICMP addon for raw access. Alternatively, an ofSystem call w/ ofProcess might be a reasonable alternative as the ping application allows non-privileged access.
    - Leverage existing Pure Poco::Net addons (e.g. ofxHTTP, ofxSMTP, ofxPOP3)
    - Others?
    - Query Network Status
    - Query NetInterface Information via Poco::Net::NetworkInterface

Tools

Project Generator

Philosophy

The Project Generator redesign includes the features of both the new and old project generators. The guiding principles are:

- Simplicity and speed for the most common use cases.
- Power for the slightly less common use cases (multiple project generation / update)
- Requiring use of the filesystem for more advanced operations.

When you first open the PG you may do one of the following:

1. Enter a project name for a new project, select a location for the project. When you hit “Generate” it will create the project file. The project location is cached by the app so that each time you restart the app it uses the same project location.
2. Drag a project file or a folder with a project file onto the app, and every field is updated. When you hit “Generate” it will update the project file(s) that correspond to the folder.
3. Drag source code or a folder with source code, but no application, and every field is updated. This includes the addon fields, which are derived from the preprocessor includes of addons.

After dragging a project file or source code, or entering a name for a new project, you may select which platforms you would like to generate project files for. By default, your current platform is selected. If the user changes the platform, say iOS, the PG will remember next time it opens.

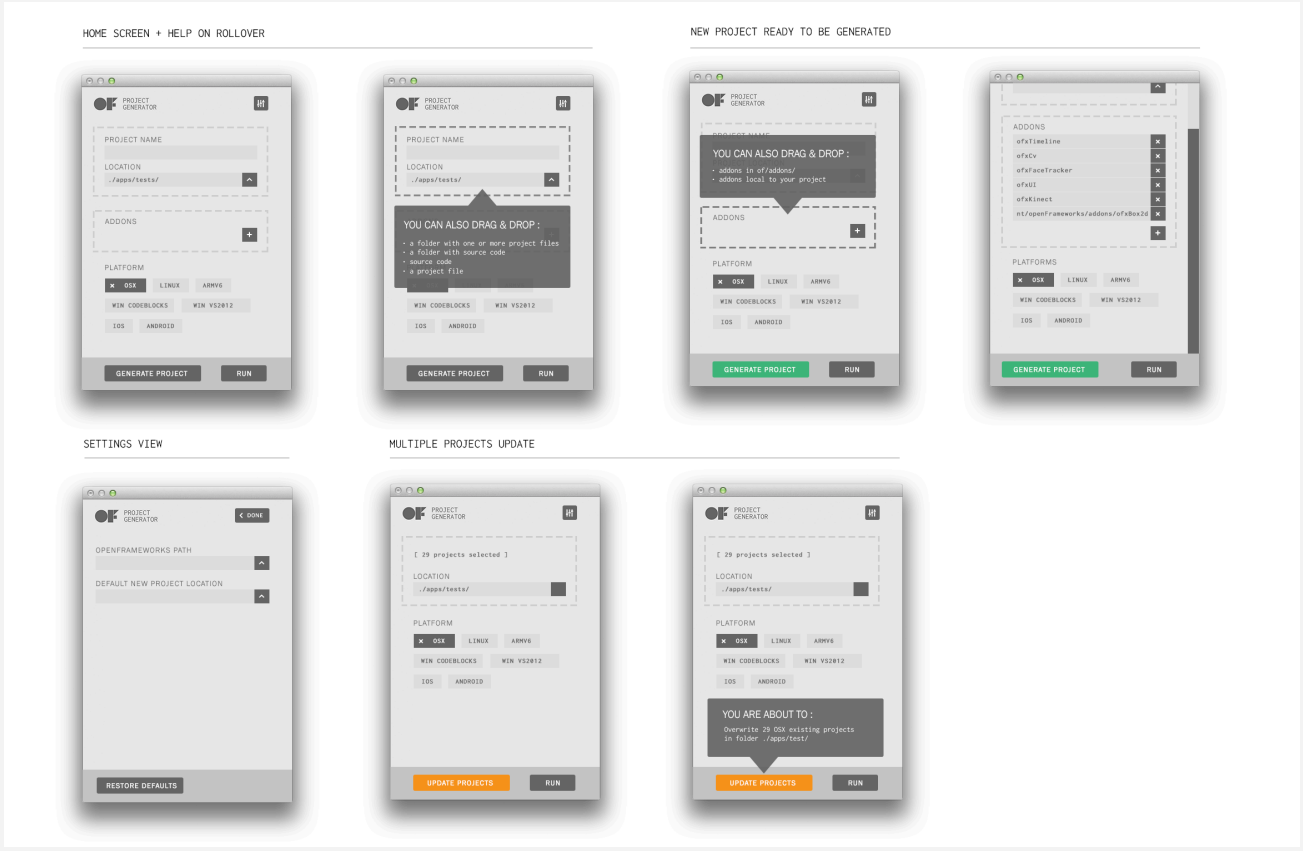
Then you can either drag addon folders into the addon region, or click a button to open a file dialog for selecting an addon. When you’re done, you can either Generate, Generate and Open, or Build and Run your project.

The entire backend to the project generator should be interfaced with through a class or set of classes, which allows us to compile a command line tool for project generation along with the project generator.

We can use the config file that ships with some addons to determine whether an addon has other addons as dependencies, and give the user the option to add those addons automatically. If the user hasn’t downloaded the addons yet, we could direct the user to GitHub or ofxAddons to download the addon.

Ideally we could use the project generator for teaching about what the process of preprocessing, compiling, and linking actually involves: what files are used where, and how they relate to each other.

Implementation



PG mockup v2, full size view, PSD (fonts available in the current PG data folder)

Implementation notes :

- 2 building blocks : a back-end app (with command line control) and a GUI app
- By default the PG will create an empty project (a name for it is required).
- Generating / Updating a project should output actions to a LOG file (track problems, debug WIP addons..etc)
- The Platform selection won't be available in released OF. Only in nightly builds and git versions (where multiple templates..Etc exist)
- The settings view modifies the PG's settings.xml file
- The [Run] button builds and runs the application using the makefile
- When updating multiple projects, the Addons section gets hidden
- zach note : we should be able to rename projects that already exist
- zach note : we should be able to parse addons into text files

Discussion

- Best way to communicate between the GUI and the backend app ?
    - "Install" the backend app via an installation script ? Could be daunting for

beginners.
    - “Install” via the GUI app ? Would require root privileges
    - Simply call the command line tool relatively from the project’s folder ?
- Removing addon on existing project will be tricky
- Instead of parsing preprocessor’s addons include, use the addons.make + addons_config.mk if available ?

Milestones

- Form a team of people interested in developing / maintaining the PG
- Section / PG Leader ?
- Implement the back-end app
    - create a specifications document that defines the command line API
    - from this document, create a diagram of the overall architecture of the program
    - create a new git repo with the github’s openFrameworks account
    - start implementing :)
- Implement the GUI app
- Implement the addon manager

Addon Generator and Manager

Philosophy

- Embedded in the PG
- Simplify projects sharing and the compilation of old project
- Handles addons dependencies
- Inspired by Ruby's bundler, web JS's Bower, Nodes' npm, Homebrew's formulas, Haxe's haxelib

Implementation

- Pure C++ implementation, no use of external scripts / tools
- Reads the project's addons.make and looks for the versions of addons used
- Check if this version is already available in the addons folder and automatically set the project/makefile's path
- Otherwise parse the addon_config.mk to find the required path :
    - if released version used : downloads & extracts an archive of the release version on github
    - if git : clone the repo and checkout the specific commit SHA
- Automatically fetched addons are cloned/downloaded in `OF_ROOT/addons/auto/`

Discussion

- What name/path for the folder where automatically fetched addons are cloned/downloaded ?
    - `OF_ROOT/addons/auto/` seems reasonable ?
- How can we add the version id or commit SHA in the addons.make file ?
    - ofxCv, 0.9.5
    - ofxCv, [url] [branch] [commit sha]
    - ofxCv, https://github.com/pizthewiz/openFrameworks.git master c691d97cb3aa133a70d721dcf0351966d9dfae9b
- We need to make sure the PG and Makefile can handle these extra informations
- Will handle addons dependencies using `ADDON_DEPENDENCIES`, which means addons developers will need to inform their addons versions in their addon_config.mk

Milestones

- Work on this could start right after the PG's back-end app is implemented

Documentation

Examples

add examples for standard c++ code, like std::vector<> and std::map<>
include videos, images, so there are some examples that don't need to be compiled to understand what they do
- Arturo wrote a nice article about vectors :
http://arturocastro.net/blog/2011/10/28/stl::vector/

Reference

- The documentation that currently lives in ofSite will be moved to openFrameworks under a documentation folder. The purpose of these markdown files is to maintain two kinds of documentation:
    1. Long form explanations of classes, and documentation that demands code snippets.
    2. Translations of the inline comments into other languages.
- The inline comments serve as the primary way of documenting OF, and they will be mixed with the markdown documentation on a per-function/method basis. When documentation is available in the markdown documents, it will be used instead of the inline documentation.

Tutorials

Overview

Tutorials should mirror the examples to a degree and fill in the conceptual details that might not immediately clear to a beginner and provide a path to learning more about openFrameworks in specific and C++ in general. They should contain their own small examples that are easy to read and focus on a single concept. Tutorials are meant to be read on the website as well as be found in the tutorials folder with the download so that new OF users have a relative wealth of resources for learning some of the basic concepts and familiarizing themselves with the structure of openFrameworks. These aren't meant to replace documentation, but augment it, and should link back to the OF documentation or other pertinent sources as much as possible to help new users to learn how to learn.

Implementation

- Maintain tutorial wishlist on website -- encourage students or new users to make them, take notice of interesting projects online and reach out for tutorials, links from community

contribute section.

- Create tutorials for:
    - How to make a tutorial (general concept, guidelines, tips)
    - Cameras
    - ofPixels / ofTextures / ofImages
    - Sound
    - gl (fbo, vbo, how to)
    - Shaders
    - Drawing - 2d primitives to ofPath and ofPolyLine
    - Events
    - std::map
    - Case studies of addons or projects

Community

openframeworks.cc

Philosophy

openframeworks.cc is a central place for people to learn about OF -- what it is, how to use it, what the community is about, and how to contribute. It should feel up to date and easy to navigate. Keep it simple -- maintain recent video reel on homepage up to date and RSS feeds working. There is a lot of energy and untapped resources (people), website should help harness and direct this energy.

The homepage should give a clear sense of what openframeworks is and what is it about, and straightforward navigation. The about page should give a philosophical overview of the mission, contributors, and supporters. The documentation page should be redesigned to allow easier navigation of documentation (see reference section for how documentation happens). The tutorials page should include links to tutorials and other resources for learning, as well as encourage and explain to others how to others create more tutorials, it should be redesigned to look more inviting and helpful. The community page should be the central place for people to get a sense of the community, to connect, and to learn how they can contribute.

Implementation

Homepage

- Add video reel.
- Internationalization: Merge all language in same ofSite repository (Jp, Kr, Ch..etc)

About

- The first section is nice but too long. Re-write first 2-3 sentences, with inline links to sections further down the page (included libs, supporters, design philosophy, faq, etc).
- Update FAQs.

Documentation

- Rename to Reference
- Remove wiki, salvage and redistribute any still relevant content.
- Remove advanced tab.
- One line at top linking to tutorial page.
- Individual class pages can link to tutorials.
- New navigation (javascript, expandable so no separate pages)
  main categories
  >classes
  >functions
- Serial + arduino should be same category?

Tutorials

- Page redesign/restructure
  - revised headings:
    - first steps (or ‘getting started) but current ‘getting started’ and ‘introduction’ tutorials should be grouped together)
    - of topics (or ‘working with OF’ to replace ‘developers’)
    - c++ concepts
    - maths
  - add thumbnails (similar to processing tutorial page), description, author
  - rework tutorial page design (similar layout to documentation, should feel like chapters in a book)

- Create and link to how to make addon tutorial from ofxaddons.
- Tutorial wishlist on website -- encourage students or new users to make them, take notice of interesting projects online and reach out for tutorials, links from community contribute section. (See tutorials section for more).

- Add section linking to books / other references (programming interactivity, c++ for beginners, nature of code).
- Update any outdated tutorials.

Community

- Setup new education (wiki) page for workshops, classes, etc (start with material from old wiki education page).
- Create and maintain short projects list -- not necessarily code related.
- This projects list can be used for GSOC -- a chance to get projects/tasks tackled, a way to build community through mentoring.

Wiki

- Delete non-github wiki, salvage still relevant content and move to of.cc.

ofxaddons.com

Philosophy

ofxAddons is a central place to discover openFrameworks addons. It should give a sense of the community and provide a platform for people to feel like their contributions will be seen. It shows the activity (heartbeat) of the community, displaying latest commits, forks, metadata, and measures of popularity and usefulness. It also serves to enforce standards to keep people formatting things in a similar way, both by maintaining an updated how-to page and by displaying examples.

Implementation

Main Page

- Users can categorize addons with hashtags in description, which will make them show up faster on addons page.
- Users can also tag addons as #unfinished or #exclude.
- Admin page includes ability to bring back addons tagged as not_addon.
- Sort addon forks by date.
- Re introduce sort by category/alphabetical

Contributors

- Map that shows users locations.
- Stargazers page -- similar to contributors page (or maybe same page) showing all users that have starred repos, which ones.

Discussion

Milestones

- Main page updates are first priority.
- Contributors page features (map, stargazers) added later.

Leaders

Philosophy

Current State: The current leader system seems to be working well, but we are missing some leaders in certain sections.

Implementation

- Visual Studio leader
- Code::Blocks leader
- New documentation leader?
- Examples leader?

Discussion

Is there enough direction for new leaders to figure out what to do? Could there be unofficial “regional / meetup leaders” that check in with outreach leader or community manager for direction / smaller projects to tackle with others in their local community?

How are leaders chosen? Maybe there should be a set term length and reevaluation at the end of that to decide if they should/want to continue or pass it on to someone else. This could allow people to look at their schedules and realistically commit for a set amount of time.

GitHub Contributions

What structural / organizational things can we change?

- all pull requestions made against master branch, with tags as the release system
- test folder for sharing expected behaviors like unit tests
- changelog updates on pull request
- milestones used to determine release dates
- make a rule against adding non-critical issues to current milestone

Continuous Integration and Testing

https://github.com/openframeworks/openFrameworks/issues/1068

- Continuous integration can be done with benben and Bruno Dias' tools. benben has setup a server that continuously builds OF on every platform. Bruno has a server that continuously merges develop into master and tests whether it compiles.
- Testing involves not just compiling examples but running and checking differences against a reference behavior.
- There is no magic bullet for testing. We just need to write the tests. This isn't specific to the 1.0 release but an ongoing goal that we will need to keep working on.