graphgrow-android

Claude Heiland-Allen

2014–2019

Contents

1 ANDROID.md

```
in Android Studio, open: example/android/uk.co.mathr.graphgrow.iface
```

2 CMakeLists.txt

```
cmake_minimum_required(VERSION 3.1.0)

project(GLFM C CXX)

option(GLFM_BUILD_EXAMPLE "Build the GLFM example" OFF)

set(CMAKE_CXX_STANDARD 11)
set(GLFM_HEADERS include/glfm.h)

if (CMAKE_SYSTEM_NAME MATCHES "Emscripten")
    set(GLFM_SRC src/glfm_platform.h src/glfm_platform_emscripten.c)
elseif (CMAKE_SYSTEM_NAME MATCHES "Android")
    set(GLFM_SRC src/glfm_platform.h src/glfm_platform_android.c ${ANDROID_NDK}/sources/android/native_app_glue/android_native_app_glue.c)
    # Set NDEBUG for android_native_app_glue to remove some superfluous logging
    set_source_files_properties(${ANDROID_NDK}/sources/android/native_app_glue/android_native_app_glue.c PROPERTIES COMPILE_FLAGS "-DNDEBUG")
else() # Assume iOS
    set(IOS TRUE)
    set(GLFM_SRC src/glfm_platform.h src/glfm_platform_ios.m)
    set(CMAKE_OSX_SYSROOT "iphoneos")
    set(GLFM_COMPILE_FLAGS "-Wno-objc-interface-ivars -Wno-objc-missing-property-synthesis -Wno-direct-ivar-access")
endif()

add_library(glfm ${GLFM_SRC} ${GLFM_HEADERS})
target_include_directories(glfm PUBLIC include)
target_include_directories(glfm PRIVATE src)

source_group(include FILES ${GLFM_HEADERS})
source_group(src FILES ${GLFM_SRC})

if (CMAKE_C_COMPILER_ID MATCHES "Clang")
    set_target_properties(glfm PROPERTIES COMPILE_FLAGS "-Weverything -Wwrite-strings -Wno-padded -Wno-covered-switch-default ${GLFM_COMPILE_FLAGS}")
elseif (CMAKE_C_COMPILER_ID MATCHES "GNU")
    set_target_properties(glfm PROPERTIES COMPILE_FLAGS "-Wall -Wextra -Wwrite-strings ${GLFM_COMPILE_FLAGS}")
elseif (CMAKE_C_COMPILER_ID MATCHES "MSVC")
    set_target_properties(glfm PROPERTIES COMPILE_FLAGS "/Wall ${GLFM_COMPILE_FLAGS}")
endif()
```

```
if (CMAKE_SYSTEM_NAME MATCHES "Android")
    find_library(log-lib log)
    find_library(android-lib android)
    find_library(EGL-lib EGL)
    find_library(GLESv2-lib GLESv2)
    target_link_libraries(glfm ${log-lib} ${android-lib} ${EGL-lib} ${GLESv2-lib
        ↘ })
    target_include_directories(glfm PRIVATE ${ANDROID_NDK}/sources/android/
        ↘ native_app_glue/)
elseif (IOS)
    target_link_libraries(glfm "-framework Foundation -framework CoreGraphics -
        ↘ framework UIKit -framework OpenGLES -framework QuartzCore")
    set_target_properties(glfm PROPERTIES
        ARCHIVE_OUTPUT_DIRECTORY ${PROJECT_BINARY_DIR}/GLFM.build/lib # For
            ↘ Archiving
        XCODE_ATTRIBUTE_SUPPORTED_PLATFORMS "iphoneos iphonesimulator appletvos
            ↘ appletvsimulator"
        XCODE_ATTRIBUTE_IPHONEOS_DEPLOYMENT_TARGET 8.0
        XCODE_ATTRIBUTE_TVOS_DEPLOYMENT_TARGET 9.0
        XCODE_ATTRIBUTE_CLANG_ENABLE_OBJC_ARC YES
    )
endif()

add_subdirectory(example)
```

3 CONTRIBUTING.md

```
# Contributing

Contributions are welcome. Thank you for making the effort.

## Fixing a Bug

Found a bug and have a fix? Great.
No need to create a different branch - just fork the repo, make the fixes, and
    ↘ create a Pull Request.

## Adding a Feature

Usually it's best to first discuss a new feature via an issue or email.

To add a feature, fork the repo, create a seperate branch, and create a Pull
    ↘ Request.
You do not have to add the feature to all three platforms (iOS, Android,
    ↘ Emscripten), but at least two platforms would be nice.
```

4 example/android/uk.co.mathr.graphgrow.iface/app/build.gradle

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 26
    defaultConfig {
        applicationId "uk.co.mathr.graphgrow.iface"
        minSdkVersion 10
        targetSdkVersion 26
```

```
        versionCode 1
        versionName "1.0"
        externalNativeBuild {
            cmake {
                arguments "-DGLFM_BUILD_EXAMPLE=ON"
            }
        }
    }
    externalNativeBuild {
        cmake {
            path "../../../../CMakeLists.txt"
        }
    }
}

android.applicationVariants.all { variant ->
    variant.outputs.all {
        outputFileName = applicationId;
        outputFileName += "-v" + android.defaultConfig.versionName;
        if (variant.buildType.name == "release") {
            outputFileName += ".apk";
        } else {
            outputFileName += "-SNAPSHOT.apk";
        }
    }
}
```

5 example/android/uk.co.mathr.graphgrow.iface/app/src/main/Android

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="uk.co.mathr.graphgrow.iface">

    <uses-feature android:glEsVersion="0x00020000" android:required="true" />
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        >
        <activity android:name="android.app.NativeActivity"
                  android:configChanges="orientation|screenLayout|screenSize|
                      ↪ keyboardHidden|keyboard">
            <meta-data
                android:name="android.app.lib_name"
                android:value="graphgrow" />
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

6 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/draw

```
<?xml version="1.0" encoding="utf-8"?>
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="108dp"
    android:height="108dp"
    android:viewportHeight="108"
    android:viewportWidth="108">
    <path
        android:fillColor="#26A69A"
        android:pathData="M0,0h108v108h-108z" />
    <path
        android:fillColor="#00000000"
        android:pathData="M9,0L9,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,0L19,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M29,0L29,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M39,0L39,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M49,0L49,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M59,0L59,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M69,0L69,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M79,0L79,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M89,0L89,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
```

```
        android:fillColor="#00000000"
        android:pathData="M99,0L99,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,9L108,9"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,19L108,19"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,29L108,29"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,39L108,39"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,49L108,49"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,59L108,59"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,69L108,69"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,79L108,79"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,89L108,89"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,99L108,99"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,29L89,29"
```

```
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,39L89,39"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,49L89,49"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,59L89,59"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,69L89,69"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,79L89,79"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M29,19L29,89"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M39,19L39,89"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M49,19L49,89"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M59,19L59,89"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M69,19L69,89"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M79,19L79,89"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
```

```
</vector>
```

7 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/draw v24/ic_launcher_foreground.xml

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:aapt="http://schemas.android.com/aapt"
    android:width="108dp"
    android:height="108dp"
    android:viewportHeight="108"
    android:viewportWidth="108">
    <path
        android:fillType="evenOdd"
        android:pathData="M32,64C32,64 38.39,52.99 44.13,50.95C51.37,48.37 ↙
            ↳ 70.14,49.57 70.14,49.57L108.26,87.69L108,109.01L75.97,107.97L32,64↙
            ↳ Z"
        android:strokeColor="#00000000"
        android:strokeWidth="1">
        <aapt:attr name="android:fillColor">
            <gradient
                android:endX="78.5885"
                android:endY="90.9159"
                android:startX="48.7653"
                android:startY="61.0927"
                android:type="linear">
                <item
                    android:color="#44000000"
                    android:offset="0.0" />
                <item
                    android:color="#00000000"
                    android:offset="1.0" />
            </gradient>
        </aapt:attr>
    </path>
    <path
        android:fillColor="#FFFFFF"
        android:fillType="nonZero"
        android:pathData="M66.94,46.02L66.94,46.02C72.44,50.07 76,56.61 76,64L32↙
            ↳ ,64C32,56.61 35.56,50.11 40.98,46.06L36.18,41.19C35.45,40.45 ↙
            ↳ 35.45,39.3 36.18,38.56C36.91,37.81 38.05,37.81 38.78,38.56L44↙
            ↳ .25,44.05C47.18,42.57 50.48,41.71 54,41.71C57.48,41.71 60.78,42.57↙
            ↳ 63.68,44.05L69.11,38.56C69.84,37.81 70.98,37.81 71.71,38.56C72↙
            ↳ .44,39.3 72.44,40.45 71.71,41.19L66.94,46.02ZM62.94,56.92C64↙
            ↳ .08,56.92 65,56.01 65,54.88C65,53.76 64.08,52.85 62.94,52.85C61↙
            ↳ .8,52.85 60.88,53.76 60.88,54.88C60.88,56.01 61.8,56.92 ↙
            ↳ 62.94,56.92ZM45.06,56.92C46.2,56.92 47.13,56.01 47.13,54.88C47↙
            ↳ .13,53.76 46.2,52.85 45.06,52.85C43.92,52.85 43,53.76 43,54.88C43↙
            ↳ ,56.01 43.92,56.92 45.06,56.92Z"
        android:strokeColor="#00000000"
        android:strokeWidth="1" />
</vector>
```

8 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/mipr anydpi-v26/ic_launcher_round.xml

```
<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/android">
    <background android:drawable="@drawable/ic_launcher_background" />
    <foreground android:drawable="@drawable/ic_launcher_foreground" />
</adaptive-icon>
```

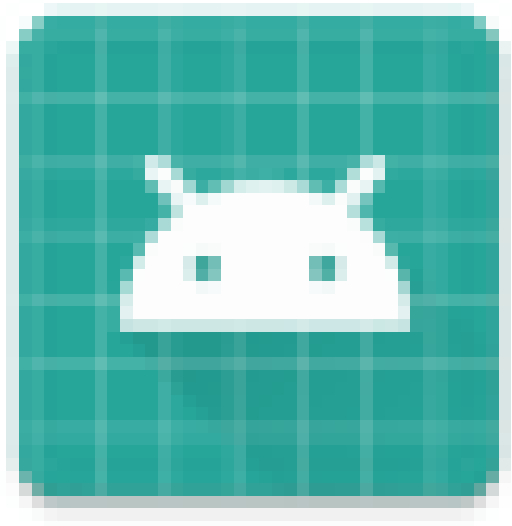
9 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/mipmap-anydpi-v26/ic_launcher.xml


```
<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/android">
    <background android:drawable="@drawable/ic_launcher_background" />
    <foreground android:drawable="@drawable/ic_launcher_foreground" />
</adaptive-icon>
```


10 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/mipmap-hdpi/ic_launcher.png

11 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/mipmap-hdpi/ic_launcher_round.png

12 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/mipmap-mdpi/ic_launcher.png



13 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/mipmap-mdpi/ic_launcher_round.png

14 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/mipmap-xhdpi/ic_launcher.png

15 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/mipmap-xhdpi/ic_launcher_round.png

16 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/mipmap-xxhdpi/ic_launcher.png

17 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/mip xxhdpi/ic_launcher_round.png

18 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/mip xxxhdpi/ic_launcher.png

19 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/mip xxxhdpi/ic_launcher_round.png

20 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/valu

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- color for the app bar and other primary UI elements -->
    <color name="colorPrimary">#3F51B5</color>

    <!-- a darker variant of the primary color, used for
         the status bar (on Android 5.0+) and contextual app bars -->
    <color name="colorPrimaryDark">#303F9F</color>

    <!-- a secondary color for controls like checkboxes and text fields -->
    <color name="colorAccent">#FF4081</color>
</resources>
```

21 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/valu

```
<resources>
    <string name="app_name">GraphGrow</string>
</resources>
```

22 example/android/uk.co.mathr.graphgrow.iface/app/src/main/res/valu

```
<resources>
  <style name="AppTheme" parent="android:Theme.Light">
  </style>
</resources>
```

23 example/android/uk.co.mathr.graphgrow.iface/build.gradle

```
buildscript {
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.1.4'
    }
}

allprojects {
    repositories {
        google()
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

24 example/android/uk.co.mathr.graphgrow.iface/.gitignore

```
.gradle
.idea
.externalNativeBuild
build
gradle
gradlew
gradlew.bat
local.properties
*.iml
```

25 example/android/uk.co.mathr.graphgrow.iface/settings.gradle

```
include ':app'
```

26 example/assets/simple.frag

```
varying lowp vec3 v_color;

void main() {
    gl_FragColor = vec4(v_color, 1.0);
}
```

27 example/assets/simple.vert

```
attribute highp vec3 a_position;
attribute lowp vec3 a_color;

varying lowp vec3 v_color;

```

```
void main() {
    gl_Position = vec4(a_position, 1.0);
    v_color = a_color;
}
```

28 example/assets/texture.frag

```
uniform lowp sampler2D texture0;

varying mediump vec2 texCoordFragment;

void main()
{
    gl_FragColor = texture2D(texture0, texCoordFragment);
}
```

29 example/assets/texture.vert

```
attribute highp vec4 position;
attribute mediump vec2 texCoord;

varying mediump vec2 texCoordFragment;

void main()
{
    gl_Position = position;
    texCoordFragment = texCoord;
}
```

30 example/cmake/GLFMAppTarget.cmake

```
#
# GLFM_APP_TARGET_NAME - App name
# GLFM_APP_ORGANIZATION_IDENTIFIER - Reverse domain name, like "com.example"
# GLFM_APP_VERSION - Version string, like "1.0"
# GLFM_APP_VERSION_ITERATION - Version code (integer)
# GLFM_APP_ASSETS_DIR - Assets directory
# GLFM_APP_SRC - Source files

file(GLOB GLFM_APP_ASSETS ${GLFM_APP_ASSETS_DIR}/*)
source_group("src" FILES ${GLFM_APP_SRC})

if (CMAKE_SYSTEM_NAME MATCHES "Emscripten")
    # HACK: Make modifications to shell_minimal.html to take up the entire browser window
    file(READ ${EMSCRIPTEN_ROOT_PATH}/src/shell_minimal.html EMSCRIPTEN_SHELL_HTML)
    string(FIND "${EMSCRIPTEN_SHELL_HTML}" "<style>" HAS_STYLE)
    if (${HAS_STYLE} EQUAL -1)
        message(WARNING "<style> not found in shell_minimal.html, copying as-is")
    else()
        string(CONCAT STYLE_REPLACEMENT "<meta name=\"viewport\" content=\"width=device-width, user-scalable=no, viewport-fit=cover\">\n"
            "    <meta name=\"apple-mobile-web-app-capable\" content=\"yes\">\n"
            "    <style>\n"
```

```
                "        /* GLFM: Start changes */\n"
                "        :root {\n"
                "            --glfm-chrome-top-old: constant(safe-area-inset-top);\n"
                "            --glfm-chrome-right-old: constant(safe-area-inset-right)
                    ↪ ;\n"
                "            --glfm-chrome-bottom-old: constant(safe-area-inset-bottom
                    ↪ );\n"
                "            --glfm-chrome-left-old: constant(safe-area-inset-left);\n
                    ↪ "
                "            --glfm-chrome-top: env(safe-area-inset-top);\n"
                "            --glfm-chrome-right: env(safe-area-inset-right);\n"
                "            --glfm-chrome-bottom: env(safe-area-inset-bottom);\n"
                "            --glfm-chrome-left: env(safe-area-inset-left);\n"
                "        }\n"
                "        body, html { border: 0px none; padding: 0px; margin: 0px;
                    ↪ width: 100%; height: 100%; overflow: hidden; position: fixed;
                    ↪ }\n"
                "        canvas.emscripten { background: black; width: 100%; height:
                    ↪ 100%; }\n"
                "        .emscripten_border { width: 100%; height: 100%; border: 0px
                    ↪ none !important;}\n"
                "        hr { display: none; }\n"
                "        /* GLFM: End changes */"
            )
            string(REPLACE "<style>" "${STYLE_REPLACEMENT}" EMSCRIPTEN_SHELL_HTML "$
                ↪ {EMSCRIPTEN_SHELL_HTML}")
        endif()
        file(WRITE ${CMAKE_CURRENT_BINARY_DIR}/shell.html.in "${
            ↪ EMSCRIPTEN_SHELL_HTML}")

        set(CMAKE_EXECUTABLE_SUFFIX ".html")
        add_executable(${GLFM_APP_TARGET_NAME} ${GLFM_APP_SRC})
        set_target_properties(${GLFM_APP_TARGET_NAME} PROPERTIES LINK_FLAGS "--shell
            ↪ -file ${CMAKE_CURRENT_BINARY_DIR}/shell.html.in --preload-file ${
            ↪ GLFM_APP_ASSETS_DIR}@")
    elseif (CMAKE_SYSTEM_NAME MATCHES "Android")
        add_library(${GLFM_APP_TARGET_NAME} SHARED ${GLFM_APP_SRC})
        target_link_libraries(${GLFM_APP_TARGET_NAME} glfm)
    else()
        # iOS. If you change this section, test archiving too.
        set(CMAKE_MACOSX_BUNDLE YES)

        add_executable(${GLFM_APP_TARGET_NAME} ${GLFM_APP_SRC} ${GLFM_APP_ASSETS})

        set_target_properties(${GLFM_APP_TARGET_NAME} PROPERTIES
            XCODE_ATTRIBUTE_PRODUCT_BUNDLE_IDENTIFIER "${
                ↪ GLFM_APP_ORGANIZATION_IDENTIFIER}.\${PRODUCT_NAME:
                ↪ rfc1034identifier}"
            XCODE_ATTRIBUTE_SUPPORTED_PLATFORMS "iphoneos iphonesimulator appletvos
                ↪ appletvsimulator"
            XCODE_ATTRIBUTE_IPHONEOS_DEPLOYMENT_TARGET 8.0          # Version
                ↪ required for GLFM
            XCODE_ATTRIBUTE_TVOS_DEPLOYMENT_TARGET 9.0
            XCODE_ATTRIBUTE_CLANG_ENABLE_OBJC_ARC YES               # ARC required
                ↪ for GLFM
            XCODE_ATTRIBUTE_TARGETED_DEVICE_FAMILY "1,2,3"          # iPhone, iPad,
                ↪ tvOS
```

```
        XCODE_ATTRIBUTE_USE_HEADERMAP YES                                  # Avoid header search paths
        XCODE_ATTRIBUTE_COMBINE_HIDPI_IMAGES NO                            # For Archiving
        XCODE_ATTRIBUTE_OTHER_LDFLAGS ""                                   # For Archiving
        XCODE_ATTRIBUTE_INSTALL_PATH "$(LOCAL_APPS_DIR)"                   # For Archiving
        XCODE_ATTRIBUTE_SKIP_INSTALL NO                                    # For Archiving
        XCODE_ATTRIBUTE_CODE_SIGN_IDENTITY "iPhone Developer"              # For convenience
    )
    set_source_files_properties(${GLFM_APP_ASSETS} LaunchScreen.storyboard PROPERTIES MACOSX_PACKAGE_LOCATION Resources)
    set_property(TARGET ${GLFM_APP_TARGET_NAME} PROPERTY MACOSX_BUNDLE_INFO_PLIST ${CMAKE_CURRENT_BINARY_DIR}/CMake-Info.plist.in)

    set(MACOSX_BUNDLE_SHORT_VERSION_STRING ${GLFM_APP_VERSION})
    set(MACOSX_BUNDLE_BUNDLE_VERSION ${GLFM_APP_VERSION_ITERATION})

    # LaunchScreen needed to allow any screen size. Don't overwrite.
    if (NOT EXISTS ${CMAKE_CURRENT_BINARY_DIR}/LaunchScreen.storyboard)
        file(WRITE ${CMAKE_CURRENT_BINARY_DIR}/LaunchScreen.storyboard
            "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?>\n"
            "<document type=\"com.apple.InterfaceBuilder3.CocoaTouch.Storyboard.XIB\" version=\"3.0\" toolsVersion=\"11134\" systemVersion=\"15F34\" targetRuntime=\"iOS.CocoaTouch\" propertyAccessControl=\"none\" useAutolayout=\"YES\" launchScreen=\"YES\" useTraitCollections=\"YES\" colorMatched=\"YES\" initialViewController=\"01J-lp-oVM\">\n"
            "    <dependencies>\n"
            "        <plugIn identifier=\"com.apple.InterfaceBuilder.IBCocoaTouchPlugin\" version=\"11106\"/>\n"
            "        <capability name=\"documents saved in the Xcode 8 format\" minToolsVersion=\"8.0\"/>\n"
            "    </dependencies>\n"
            "    <scenes>\n"
            "        <!--View Controller-->\n"
            "        <scene sceneID=\"EHf-IW-A2E\">\n"
            "            <objects>\n"
            "                <viewController id=\"01J-lp-oVM\" sceneMemberID=\"viewController\">\n"
            "                    <layoutGuides>\n"
            "                        <viewControllerLayoutGuide type=\"top\" id=\"Llm-lL-Icb\"/>\n"
            "                        <viewControllerLayoutGuide type=\"bottom\" id=\"xb3-aO-Qok\"/>\n"
            "                    </layoutGuides>\n"
            "                    <view key=\"view\" contentMode=\"scaleToFill\" id=\"Ze5-6b-2t3\">\n"
            "                        <rect key=\"frame\" x=\"0.0\" y=\"0.0\" width=\"375\" height=\"667\"/>\n"
            "                        <autoresizingMask key=\"autoresizingMask\" widthSizable=\"YES\" heightSizable=\"YES\"/>\n"
            "                        <color key=\"backgroundColor\" red=\"0\" green=\"0\" blue=\"0\" alpha=\"1\" colorSpace=\"custom\" customColorSpace=\"sRGB\"/>\n"
            "                    </view>\n"
            "                </viewController>\n"
```

```
                "                    <placeholder placeholderIdentifier=\"↙
                    ↳ IBFirstResponder\" id=\"iYj-Kq-Ea1\" userLabel=\"First ↙
                    ↳ Responder\" sceneMemberID=\"firstResponder\"/>\n"
100             "                </objects>\n"
                "                <point key=\"canvasLocation\" x=\"53\" y=\"375\"/>\n"
                "            </scene>\n"
                "    </scenes>\n"
                "</document>\n"
105         )
        endif()
        # In place of MacOSXBundleInfo.plist.in
        file(WRITE ${CMAKE_CURRENT_BINARY_DIR}/CMake-Info.plist.in
            "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n"
110         "<!DOCTYPE plist PUBLIC \"-//Apple//DTD PLIST 1.0//EN" "http://www.apple↙
                ↳ .com/DTDs/PropertyList-1.0.dtd\">\n"
            "<plist version=\"1.0\">\n"
            "<dict>\n"
            "        <key>CFBundleDevelopmentRegion</key>\n"
            "        <string>en</string>\n"
115         "        <key>CFBundleExecutable</key>\n"
            "        <string>$(EXECUTABLE_NAME)</string>\n"
            "        <key>CFBundleIdentifier</key>\n"
            "        <string>$(PRODUCT_BUNDLE_IDENTIFIER)</string>\n"
            "        <key>CFBundleInfoDictionaryVersion</key>\n"
120         "        <string>6.0</string>\n"
            "        <key>CFBundleName</key>\n"
            "        <string>$(PRODUCT_NAME)</string>\n"
            "        <key>CFBundlePackageType</key>\n"
            "        <string>APPL</string>\n"
125         "        <key>CFBundleShortVersionString</key>\n"
            "        <string>\${MACOSX_BUNDLE_SHORT_VERSION_STRING}</string>\n"
            "        <key>CFBundleVersion</key>\n"
            "        <string>\${MACOSX_BUNDLE_BUNDLE_VERSION}</string>\n"
            "        <key>LSRequiresIPhoneOS</key>\n"
130         "        <true/>\n"
            "        <key>UILaunchStoryboardName</key>\n"
            "        <string>LaunchScreen</string>\n"
            "        <key>UIRequiredDeviceCapabilities</key>\n"
            "        <array>\n"
135         "                <string>armv7</string>\n"
            "                <string>opengles-2</string>\n"
            "        </array>\n"
            "        <key>UIStatusBarHidden</key>\n"
            "        <true/>\n"
140         "        <key>UISupportedInterfaceOrientations</key>\n"
            "        <array>\n"
            "                <string>UIInterfaceOrientationPortrait</string>\n"
            "                <string>UIInterfaceOrientationLandscapeLeft</string>\n"
            "                <string>UIInterfaceOrientationLandscapeRight</string>\n"
145         "        </array>\n"
            "        <key>UISupportedInterfaceOrientations~ipad</key>\n"
            "        <array>\n"
            "                <string>UIInterfaceOrientationPortrait</string>\n"
            "                <string>UIInterfaceOrientationPortraitUpsideDown</string ↙
                ↳ >\n"
150         "                <string>UIInterfaceOrientationLandscapeLeft</string>\n"
            "                <string>UIInterfaceOrientationLandscapeRight</string>\n"
```

```
        "        </array>\n"
        "</dict>\n"
        "</plist>\n"
    )
endif()
```

31 example/CMakeLists.txt

```
cmake_minimum_required(VERSION 3.6.0) # Might run on earlier versions. Probably ↙
    ↪ requires 3.4 or 3.5

link_libraries(glfm)
list(APPEND CMAKE_MODULE_PATH "${CMAKE_CURRENT_LIST_DIR}/cmake")

# Common
set(GLFM_APP_ORGANIZATION_IDENTIFIER "uk.co.mathr")
set(GLFM_APP_VERSION "1.0")
set(GLFM_APP_VERSION_ITERATION 1)
set(GLFM_APP_ASSETS_DIR ${CMAKE_CURRENT_SOURCE_DIR}/assets)

# Main example

set(lo_DIR liblo-0.29)

set(GLFM_APP_TARGET_NAME graphgrow)

set(GLFM_APP_SRC
    src/graphgrow-iface.cpp
    src/file_compat.h
    lo/lo.h
    lo/lo_endian.h
    ${lo_DIR}/src/address.c
    ${lo_DIR}/src/blob.c
    ${lo_DIR}/src/bundle.c
    ${lo_DIR}/src/message.c
    ${lo_DIR}/src/method.c
    ${lo_DIR}/src/pattern_match.c
    ${lo_DIR}/src/send.c
    ${lo_DIR}/src/server.c
    ${lo_DIR}/src/timetag.c
    ${lo_DIR}/lo/lo_cpp.h
    ${lo_DIR}/lo/lo_errors.h
    ${lo_DIR}/lo/lo_lowlevel.h
    ${lo_DIR}/lo/lo_macros.h
    ${lo_DIR}/lo/lo_osc_types.h
    ${lo_DIR}/lo/lo_serverthread.h
    ${lo_DIR}/lo/lo_throw.h
    ${lo_DIR}/lo/lo_types.h
    )

add_definitions(-DPACKAGE_NAME="liblo")
add_definitions(-DPRINTF_LL="ll")
add_definitions(-DHAVE_POLL)
add_definitions(-DHAVE_SELECT)

include_directories(./ ${lo_DIR})
```

```
include(GLFMAppTarget)
```

32 example/lo/lo_endian.h

```
/*
 *  Copyright (C) 2014 Steve Harris et al. (see AUTHORS)
 *
 *  This program is free software; you can redistribute it and/or
 *  modify it under the terms of the GNU Lesser General Public License
 *  as published by the Free Software Foundation; either version 2.1
 *  of the License, or (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU Lesser General Public License for more details.
 *
 *  $Id$
 */

#ifndef LO_ENDIAN_H
#define LO_ENDIAN_H

#include <sys/types.h>

#ifdef _MSC_VER
#ifndef UINTSDEFINED
#define UINTSDEFINED
#define int32_t __int32
#define int64_t __int64
#define uint32_t unsigned __int32
#define uint64_t unsigned __int64
#define uint8_t unsigned __int8
#endif
#else
#include <stdint.h>
#endif

#if defined(WIN32) || defined(_MSC_VER)
#include <winsock2.h>
#include <ws2tcpip.h>
#else
#include <netinet/in.h>
#endif

#ifdef __cplusplus
extern "C" {
#endif

#define lo_swap16(x) htons(x)

#define lo_swap32(x) htonl(x)

/* These macros come from the Linux kernel */

#ifndef lo_swap16
#define lo_swap16(x) \
```

```
    ({ \
55      uint16_t __x = (x); \
        ((uint16_t)( \
            (((uint16_t)(__x) & (uint16_t)0x00ffU) << 8) | \
            (((uint16_t)(__x) & (uint16_t)0xff00U) >> 8) )); \
    })
60  #warning USING UNOPTIMISED ENDIAN STUFF
    #endif

    #ifndef lo_swap32
    #define lo_swap32(x) \
65  ({ \
        uint32_t __x = (x); \
        ((uint32_t)( \
            (((uint32_t)(__x) & (uint32_t)0x000000ffUL) << 24) | \
            (((uint32_t)(__x) & (uint32_t)0x0000ff00UL) <<  8) | \
70          (((uint32_t)(__x) & (uint32_t)0x00ff0000UL) >>  8) | \
            (((uint32_t)(__x) & (uint32_t)0xff000000UL) >> 24) )); \
    })
    #endif

75  #if 0
    #ifndef lo_swap64
    #define lo_swap64(x) \
    ({ \
        uint64_t __x = (x); \
80      ((uint64_t)( \
            (uint64_t)(((uint64_t)(__x) & (uint64_t)0x00000000000000ffULL) << 56) | ↙
                ↳ \
            (uint64_t)(((uint64_t)(__x) & (uint64_t)0x000000000000ff00ULL) << 40) | ↙
                ↳ \
            (uint64_t)(((uint64_t)(__x) & (uint64_t)0x0000000000ff0000ULL) << 24) | ↙
                ↳ \
            (uint64_t)(((uint64_t)(__x) & (uint64_t)0x00000000ff000000ULL) <<  8) | ↙
                ↳ \
85          (uint64_t)(((uint64_t)(__x) & (uint64_t)0x000000ff00000000ULL) >>  8) | ↙
                ↳ \
            (uint64_t)(((uint64_t)(__x) & (uint64_t)0x0000ff0000000000ULL) >> 24) | ↙
                ↳ \
            (uint64_t)(((uint64_t)(__x) & (uint64_t)0x00ff000000000000ULL) >> 40) | ↙
                ↳ \
            (uint64_t)(((uint64_t)(__x) & (uint64_t)0xff00000000000000ULL) >> 56) ))↙
                ↳ ; \
    })
90  #endif
    #else

    typedef union {
        uint64_t all;
95      struct {
            uint32_t a;
            uint32_t b;
        } part;
    } lo_split64;
100
    #ifdef _MSC_VER
    #define LO_INLINE __inline
```

```
#else
#define LO_INLINE inline
#endif
static LO_INLINE uint64_t lo_swap64(uint64_t x)
{
    lo_split64 in, out;

    in.all = x;
    out.part.a = lo_swap32(in.part.b);
    out.part.b = lo_swap32(in.part.a);

    return out.all;
}
#undef LO_INLINE
#endif

#ifdef LO_BIGENDIAN
#undef LO_BIGENDIAN
#endif

#ifdef __BIG_ENDIAN__
#define LO_BIGENDIAN 1
#else
#ifdef __LITTLE_ENDIAN__
#define LO_BIGENDIAN 0
#else
#error Unknown endianness
#endif
#endif

/* Host to OSC and OSC to Host conversion macros */
#if LO_BIGENDIAN
#define lo_htoo16(x) (x)
#define lo_htoo32(x) (x)
#define lo_htoo64(x) (x)
#define lo_otoh16(x) (x)
#define lo_otoh32(x) (x)
#define lo_otoh64(x) (x)
#else
#define lo_htoo16 lo_swap16
#define lo_htoo32 lo_swap32
#define lo_htoo64 lo_swap64
#define lo_otoh16 lo_swap16
#define lo_otoh32 lo_swap32
#define lo_otoh64 lo_swap64
#endif

#ifdef __cplusplus
}
#endif

#endif

/* vi:set ts=8 sts=4 sw=4: */
```

33 example/lo/lo.h

```
/*
 *  Copyright (C) 2014 Steve Harris et al. (see AUTHORS)
 *
 *  This program is free software; you can redistribute it and/or
 *  modify it under the terms of the GNU Lesser General Public License
 *  as published by the Free Software Foundation; either version 2.1
 *  of the License, or (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU Lesser General Public License for more details.
 *
 *  $Id$
 */

#ifndef LO_H
#define LO_H

#ifdef __cplusplus
extern "C" {
#endif

/**
 * \file lo.h The liblo main headerfile and high-level API functions.
 */

#include "lo/lo_endian.h"
#include "lo/lo_types.h"
#include "lo/lo_osc_types.h"
#include "lo/lo_errors.h"
#include "lo/lo_lowlevel.h"

/**
 * \defgroup liblo High-level OSC API
 *
 * Defines the high-level API functions necessary to implement OSC support.
 * Should be adequate for most applications, but if you require lower level
 * control you can use the functions defined in lo_lowlevel.h
 * @{
 */

/**
 * \brief Declare an OSC destination, given IP address and port number.
 * Same as lo_address_new_with_proto(), but using UDP.
 *
 * \param host An IP address or number, or NULL for the local machine.
 * \param port a decimal port number or service name.
 *
 * The lo_address object may be used as the target of OSC messages.
 *
 * Note: if you wish to receive replies from the target of this
 * address, you must first create a lo_server_thread or lo_server
 * object which will receive the replies. The last lo_server(_thread)
 * object craeted will be the receiver.
 */
lo_address lo_address_new(const char *host, const char *port);
```

```
    /**
60   * \brief Declare an OSC destination, given IP address and port number,
     * specifying protocol.
     *
     * \param proto The protocol to use, must be one of LO_UDP, LO_TCP or LO_UNIX.
     * \param host An IP address or number, or NULL for the local machine.
65   * \param port a decimal port number or service name.
     *
     * The lo_address object may be used as the target of OSC messages.
     *
     * Note: if you wish to receive replies from the target of this
70   * address, you must first create a lo_server_thread or lo_server
     * object which will receive the replies. The last lo_server(_thread)
     * object created will be the receiver.
     */
    lo_address lo_address_new_with_proto(int proto, const char *host, const char *↙
        ↳ port);
75
    /**
     * \brief Create a lo_address object from an OSC URL.
     *
     * example: \c "osc.udp://localhost:4444/my/path/"
80   */
    lo_address lo_address_new_from_url(const char *url);

    /**
     * \brief Free the memory used by the lo_address object
85   */
    void lo_address_free(lo_address t);

    /**
     * \brief Set the Time-to-Live value for a given target address.
90   *
     * This is required for sending multicast UDP messages.  A value of 1
     * (the usual case) keeps the message within the subnet, while 255
     * means a global, unrestricted scope.
     *
95   * \param t An OSC address.
     * \param ttl An integer specifying the scope of a multicast UDP message.
     */
    void lo_address_set_ttl(lo_address t, int ttl);

100 /**
     * \brief Get the Time-to-Live value for a given target address.
     *
     * \param t An OSC address.
     * \return An integer specifying the scope of a multicast UDP message.
105  */
    int lo_address_get_ttl(lo_address t);

    /**
     * \brief Send a OSC formatted message to the address specified.
110  *
     * \param targ The target OSC address
     * \param path The OSC path the message will be delivered to
     * \param type The types of the data items in the message, types are defined in
```

```
     * lo_osc_types.h
115  * \param ... The data values to be transmitted. The types of the arguments
     * passed here must agree with the types specified in the type parameter.
     *
     * example:
     * \code
120  * lo_send(t, "/foo/bar", "ff", 0.1f, 23.0f);
     * \endcode
     *
     * \return -1 on failure.
     */
125 int lo_send(lo_address targ, const char *path, const char *type, ...);

    /**
     * \brief Send a OSC formatted message to the address specified,
     * from the same socket as the specified server.
130  *
     * \param targ The target OSC address
     * \param from The server to send message from   (can be NULL to use new socket)
     * \param ts   The OSC timetag timestamp at which the message will be processed
     * (can be LO_TT_IMMEDIATE if you don't want to attach a timetag)
135  * \param path The OSC path the message will be delivered to
     * \param type The types of the data items in the message, types are defined in
     * lo_osc_types.h
     * \param ... The data values to be transmitted. The types of the arguments
     * passed here must agree with the types specified in the type parameter.
140  *
     * example:
     * \code
     * serv = lo_server_new(NULL, err);
     * lo_server_add_method(serv, "/reply", "ss", reply_handler, NULL);
145  * lo_send_from(t, serv, LO_TT_IMMEDIATE, "/foo/bar", "ff", 0.1f, 23.0f);
     * \endcode
     *
     * \return on success, the number of bytes sent, or -1 on failure.
     */
150 int lo_send_from(lo_address targ, lo_server from, lo_timetag ts,
                         const char *path, const char *type, ...);

    /**
     * \brief Send a OSC formatted message to the address specified, scheduled to
155  * be dispatch at some time in the future.
     *
     * \param targ The target OSC address
     * \param ts The OSC timetag timestamp at which the message will be processed
     * \param path The OSC path the message will be delivered to
160  * \param type The types of the data items in the message, types are defined in
     * lo_osc_types.h
     * \param ... The data values to be transmitted. The types of the arguments
     * passed here must agree with the types specified in the type parameter.
     *
165  * example:
     * \code
     * lo_timetag now;<br>
     * lo_timetag_now(&now);<br>
     * lo_send_timestamped(t, now, "/foo/bar", "ff", 0.1f, 23.0f);
170  * \endcode
```

```
     *
     * \return on success, the number of bytes sent, or -1 on failure.
     */
    int lo_send_timestamped(lo_address targ, lo_timetag ts, const char *path,
175                         const char *type, ...);

    /**
     * \brief Return the error number from the last failed lo_send() or
     * lo_address_new() call
180  */
    int lo_address_errno(lo_address a);

    /**
     * \brief Return the error string from the last failed lo_send() or
185  * lo_address_new() call
     */
    const char *lo_address_errstr(lo_address a);

    /**
190  * \brief Create a new OSC blob type.
     *
     * \param size The amount of space to allocate in the blob structure.
     * \param data The data that will be used to initialise the blob, should be
     * size bytes long.
195  */
    lo_blob lo_blob_new(int32_t size, const void *data);

    /**
     * \brief Free the memory taken by a blob
200  */
    void lo_blob_free(lo_blob b);

    /**
     * \brief Return the amount of valid data in a lo_blob object.
205  *
     * If you want to know the storage size, use lo_arg_size().
     */
    uint32_t lo_blob_datasize(lo_blob b);

210 /**
     * \brief Return a pointer to the start of the blob data to allow contents to
     * be changed.
     */
    void *lo_blob_dataptr(lo_blob b);
215
    /**
     * \brief Get information on the version of liblo current in use.
     *
     * All parameters are optional and can be given the value of 0 if that
220  * information is not desired.  For example, to get just the version
     * as a string, call lo_version(str, size, 0, 0, 0, 0, 0, 0, 0);
     *
     * The "lt" fields, called the ABI version, corresponds to libtool's
     * versioning system for binary interface compatibility, and is not
225  * related to the library version number.  This information is usually
     * encoded in the filename of the shared library.
     *
```

```
     * Typically the string returned in 'verstr' should correspond with
     * $major.$minor$extra, e.g., "0.28rc".  If no 'extra' information is
230  * present, e.g., "0.28", extra will given the null string.
     *
     * \param verstr       A buffer to receive a string describing the
     *                     library version.
     * \param verstr_size  Size of the buffer pointed to by string.
235  * \param major        Location to receive the library major version.
     * \param minor        Location to receive the library minor version.
     * \param extra        Location to receive the library version extra string.
     * \param extra_size   Size of the buffer pointed to by extra.
     * \param lt_major     Location to receive the ABI major version.
240  * \param lt_minor     Location to receive the ABI minor version.
     * \param lt_bug       Location to receive the ABI 'bugfix' version.
     */
    void lo_version(char *verstr, int verstr_size,
                    int *major, int *minor, char *extra, int extra_size,
245                 int *lt_major, int *lt_minor, int *lt_bug);

    /** @} */

    #include "lo/lo_macros.h"
250
    #ifdef __cplusplus
    }
    #endif

255 #endif
```

34 example/src/file_compat.h

```
    /*
     file-compat
     https://github.com/brackeen/file-compat
     Copyright (c) 2017 David Brackeen
5
     Permission is hereby granted, free of charge, to any person obtaining a copy of↙
         ↳ this software and
     associated documentation files (the "Software"), to deal in the Software ↙
         ↳ without restriction,
     including without limitation the rights to use, copy, modify, merge, publish, ↙
         ↳ distribute,
     sublicense, and/or sell copies of the Software, and to permit persons to whom ↙
         ↳ the Software is
10   furnished to do so, subject to the following conditions:

     The above copyright notice and this permission notice shall be included in all ↙
         ↳ copies or
     substantial portions of the Software.

15   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR ↙
         ↳ IMPLIED, INCLUDING BUT
     NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR ↙
         ↳ PURPOSE AND
     NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE ↙
         ↳ FOR ANY CLAIM,
     DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE↙
```

```
    ↪ , ARISING FROM, OUT
 OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE ↙
    ↪ SOFTWARE.
 */

#ifndef FILE_COMPAT_H
#define FILE_COMPAT_H

/**
    Redefines common `stdio` functions so that they work as expected on Windows ↙
        ↪ and Android.

    Additionally, includes the function `fc_resdir() to get the path to the ↙
        ↪ current executable's
    directory or its resources directory. This function works on Windows, Linux,↙
        ↪  macOS, iOS, and
    Android.

    | Function             | Windows                          | Android
    |----------------------|----------------------------------|-----------------------------------
        ↪
    | `printf`             | Uses `OutputDebugString`*        | Uses `↙
        ↪ __android_log_print`
    | `fopen`              | Uses `fopen_s`                   | Uses `↙
        ↪ AAssetManager_open` if read mode
    | `fclose`             | Adds `NULL` check                | No change
    | `chdir`              | Uses `_chdir`                    | No change
    | `sleep` / `usleep`   | Uses `Sleep`                     | No change

    *`OutputDebugString` is only used if the debugger is present and no console ↙
        ↪ is allocated.
    Otherwise uses `printf`.

    ## Usage
    For Android, define `FILE_COMPAT_ANDROID_ACTIVITY` to be a reference to an `↙
        ↪ ANativeActivity`
    instance or to a function that returns an `ANativeActivity` instance. May be↙
        ↪  `NULL`.

        #define FILE_COMPAT_ANDROID_ACTIVITY app->activity
        #include "file_compat.h"
 */

#include <stdio.h>
#include <errno.h>
#if defined(_WIN32)
#  include <direct.h>
#  if !defined(PATH_MAX)
#    define PATH_MAX 1024
#  endif
#else
#  include <unistd.h>
#  include <limits.h> /* PATH_MAX */
#endif
#if defined(__APPLE__)
#  include <CoreFoundation/CoreFoundation.h>
#endif
```

```
65
    /**
        Gets the path to the current executable's resources directory. On macOS/iOS, ↙
            ↳ this is the path to
        the bundle's resources. On Windows and Linux, this is a path to the ↙
            ↳ executable's directory.
        On Android and Emscripten, this is an empty string.
70
        The path will have a trailing slash (or backslash on Windows), except for ↙
            ↳ the empty strings for
        Android and Emscripten.

        @param path The buffer to fill the path. No more than `path_max` bytes are ↙
            ↳ writen to the buffer,
75      including the trailing 0. If failure occurs, the path is set to an empty ↙
            ↳ string.
        @param path_max The length of the buffer. Should be `PATH_MAX`.
        @return 0 on success, -1 on failure.
     */
    static int fc_resdir(char *path, size_t path_max) {
80      if (!path || path_max == 0) {
            return -1;
        }
    #if defined(_WIN32)
        DWORD length = GetModuleFileNameA(NULL, path, path_max);
85      if (length > 0 && length < path_max) {
            for (DWORD i = length - 1; i > 0; i--) {
                if (path[i] == '\\') {
                    path[i + 1] = 0;
                    return 0;
90              }
            }
        }
        path[0] = 0;
        return -1;
95  #elif defined(__linux__) && !defined(__ANDROID__)
        ssize_t length = readlink("/proc/self/exe", path, path_max - 1);
        if (length > 0 && length < path_max) {
            for (ssize_t i = length - 1; i > 0; i--) {
                if (path[i] == '/') {
100                 path[i + 1] = 0;
                    return 0;
                }
            }
        }
105     path[0] = 0;
        return -1;
    #elif defined(__APPLE__)
        CFBundleRef bundle = CFBundleGetMainBundle();
        if (bundle) {
110         CFURLRef resourcesURL = CFBundleCopyResourcesDirectoryURL(bundle);
            if (resourcesURL) {
                Boolean success = CFURLGetFileSystemRepresentation(resourcesURL, ↙
                    ↳ TRUE, (UInt8 *)path,
                                                                   path_max - 1);
                CFRelease(resourcesURL);
115             if (success) {
```

```
                unsigned long length = strlen(path);
                if (length > 0 && length < path_max - 1) {
                    // Add trailing slash
                    if (path[length - 1] != '/') {
                        path[length] = '/';
                        path[length + 1] = 0;
                    }
                    return 0;
                }
            }
        }
    }
    path[0] = 0;
    return -1;
#elif defined(__ANDROID__)
    path[0] = 0;
    return 0;
#elif defined(__EMSCRIPTEN__)
    path[0] = 0;
    return 0;
#else
#error Unsupported platform
#endif
}

/* MARK: Windows */

#if defined(_WIN32)

static inline unsigned int sleep(unsigned int seconds) {
    Sleep(seconds * 1000);
    return 0;
}

static inline int usleep(unsigned long useconds) {
    if (useconds >= 1000000) {
        errno = EINVAL;
        return -1;
    } else {
        Sleep(useconds / 1000);
        return 0;
    }
}

static inline FILE *_fc_windows_fopen(const char *filename, const char *mode) {
    FILE *file = NULL;
    fopen_s(&file, filename, mode);
    return file;
}

static inline int _fc_windows_fclose(FILE *stream) {
    // The Windows fclose() function will crash if stream is NULL
    if (stream) {
        return fclose(stream);
    } else {
        return 0;
    }
```

```
}

#define fopen(filename, mode) _fc_windows_fopen(filename, mode)
#define fclose(file) _fc_windows_fclose(file)
#define chdir(dirname) _chdir(dirname)

#if defined(_DEBUG)

// Outputs to debug window if there is no console and IsDebuggerPresent() ↙
    ↳ returns true.
static int _fc_printf(const char *format, ...) {
    int result;
    if (IsDebuggerPresent() && GetStdHandle(STD_OUTPUT_HANDLE) == NULL) {
        char buffer[1024];
        va_list args;
        va_start(args, format);
        result = vsprintf_s(buffer, sizeof(buffer), format, args);
        va_end(args);
        if (result >= 0) {
            OutputDebugStringA(buffer);
        }
    } else {
        va_list args;
        va_start(args, format);
        result = vprintf(format, args);
        va_end(args);
    }
    return result;
}

#define printf(format, ...) _fc_printf(format, __VA_ARGS__)

#endif /* _DEBUG */

#endif /* _WIN32 */

/* MARK: Android */

#if defined(__ANDROID__)

#if !defined(_BSD_SOURCE)
FILE* funopen(const void* __cookie,
              int (*__read_fn)(void*, char*, int),
              int (*__write_fn)(void*, const char*, int),
              fpos_t (*__seek_fn)(void*, fpos_t, int),
              int (*__close_fn)(void*));
#endif /* _BSD_SOURCE */

#if !defined(FILE_COMPAT_ANDROID_ACTIVITY)
#error FILE_COMPAT_ANDROID_ACTIVITY must be defined as a reference to an ↙
    ↳ ANativeActivity (or NULL).
#endif

#include <android/log.h>

static int _fc_android_read(void *cookie, char *buf, int size) {
    return AAsset_read((AAsset *)cookie, buf, (size_t)size);
```

```
}

static int _fc_android_write(void *cookie, const char *buf, int size) {
    (void)cookie;
    (void)buf;
    (void)size;
    errno = EACCES;
    return -1;
}

static fpos_t _fc_android_seek(void *cookie, fpos_t offset, int whence) {
    return AAsset_seek((AAsset *)cookie, offset, whence);
}

static int _fc_android_close(void *cookie) {
    AAsset_close((AAsset *)cookie);
    return 0;
}

static FILE *_fc_android_fopen(const char *filename, const char *mode) {
    ANativeActivity *activity = FILE_COMPAT_ANDROID_ACTIVITY;
    AAssetManager *assetManager = NULL;
    AAsset *asset = NULL;
    if (activity) {
        assetManager = activity->assetManager;
    }
    if (assetManager && mode && mode[0] == 'r') {
        asset = AAssetManager_open(assetManager, filename, AASSET_MODE_UNKNOWN);
    }
    if (asset) {
        return funopen(asset, _fc_android_read, _fc_android_write, ↙
            ↳ _fc_android_seek,
                       _fc_android_close);
    } else {
        return fopen(filename, mode);
    }
}

#define printf(...) __android_log_print(ANDROID_LOG_INFO, "stdout", __VA_ARGS__)
#define fopen(filename, mode) _fc_android_fopen(filename, mode)

#endif /* __ANDROID__ */

#endif /* FILE_COMPAT_H */
```

35 example/src/graphgrow-iface.cpp

```
#define HAVE_LO

#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <pthread.h>
#include <unistd.h>

#include <vector>
```

```
#include "glfm.h"
#define FILE_COMPAT_ANDROID_ACTIVITY glfmAndroidGetActivity()
#include "file_compat.h"

#include "/usr/include/glm/glm.hpp"

#ifdef HAVE_LO
#include "lo/lo.h"
#endif


static PFNGLGENVERTEXARRAYSOESPROC glGenVertexArraysOES;
static PFNGLBINDVERTEXARRAYOESPROC glBindVertexArrayOES;
static PFNGLDELETEVERTEXARRAYSOESPROC glDeleteVertexArraysOES;
static PFNGLISVERTEXARRAYOESPROC glIsVertexArrayOES;
#define glGenVertexArrays glGenVertexArraysOES
#define glBindVertexArray glBindVertexArrayOES
#define glDeleteVertexArrays glDeleteVertexArraysOES
#define glIsVertexArray glIsVertexArrayOES
#define eglGetProcAddress glfmGetProcAddress

// must match audio and video implementations
// NOTE: define both to 8 when using full GUI
// otherwise breakage occurs
#define GG_AUDIO_LINKS 1
#define GG_VIDEO_LINKS 4

// ----------------------------- declarations -------------------------

struct video_control;
struct audio_control;

enum graph_mode
{
  mode_create_node,
  mode_delete_node,
  mode_move_node,
  mode_create_link,
  mode_delete_link,
  mode_flip_link,
  mode_target_link,
  mode_help
};

struct graph_node;
struct graph_link;
struct graph_arrow;
struct graph_cord;
struct graph_designer;

typedef bool (*node_mouse_t)(graph_node *, double, double);
typedef bool (*link_mouse_t)(graph_link *, double, double);
typedef bool (*arrow_mouse_t)(graph_arrow *, double, double);
typedef bool (*cord_mouse_t)(graph_cord *, double, double);
typedef bool (*designer_mouse_t)(graph_designer *, double, double);
```

```
graph_link *link_new(graph_designer *designer, graph_node *from, graph_node *to)↙
    ↳ ;
void link_delete(graph_link *link);
void link_leave_mode(graph_link *link);
void link_enter_mode(graph_link *link, graph_mode mode);
void link_update(graph_link *link);
graph_arrow *arrow_new();
void arrow_delete(graph_arrow *arrow);
void arrow_leave_mode(graph_arrow *arrow);
void arrow_enter_mode(graph_arrow *arrow, graph_mode mode);
graph_cord *cord_new(double x1, double y1, double x2, double y2);
void cord_delete(graph_cord *cord);
graph_node *node_new(graph_designer *designer, double x, double y, bool fixed);
void node_delete(graph_node *node);
bool nodes_linked(graph_node *a, graph_node *b);
void node_leave_mode(graph_node *node);
bool node_mouse_hit(graph_node *node, double x, double y);
bool node_mouse_down_new(graph_node *node, double x, double y);
bool node_mouse_down_delete(graph_node *node, double x, double y);
bool node_mouse_down_move(graph_node *node, double x, double y);
bool node_mouse_down_link(graph_node *node, double x, double y);
bool node_mouse_up_corded(graph_node *node, double x, double y);
void node_enter_mode(graph_node *node, graph_mode mode);
graph_designer *designer_new(int id);
void designer_leave_mode(graph_designer *designer);
void designer_enter_mode(graph_designer *designer, graph_mode mode);
void node_move_by(graph_node *node, double dx, double dy);
bool designer_mouse_down(graph_designer *designer, double x, double y);
bool designer_mouse_down_node_new(graph_designer *designer, double x, double y);
bool designer_mouse_move_dragging(graph_designer *designer, double x, double y);
bool designer_mouse_move_dragged(graph_designer *designer, double x, double y);
bool designer_mouse_move_cording(graph_designer *designer, double x, double y);
bool designer_mouse_move_corded(graph_designer *designer, double x, double y);

// ---------------------------- constants ----------------------------

const glm::vec3 white = glm::vec3(1.0f);
const glm::vec3 black = glm::vec3(0.0f);
const glm::vec3 colours[4] =
  { glm::vec3(1.000f, 0.800f, 0.000f)
  , glm::vec3(0.125f, 1.000f, 0.250f)
  , glm::vec3(0.125f, 0.533f, 1.000f)
  , glm::vec3(1.000f, 0.125f, 0.800f)
  };

static int screen_width = 1280;
static int screen_height = 800;
static int screen_border = 20;

const double minimum_x = 50;
const double minimum_y = 50;
const double maximum_x = 750;
const double maximum_y = 750;

// ---------------------------- controls ----------------------------

struct video_control
```

```
{
  int32_t active;
  int32_t count    [4];
  int32_t source   [4][GG_VIDEO_LINKS];
  float   scale    [4][GG_VIDEO_LINKS];
  float   transform[4][GG_VIDEO_LINKS][3][3];
  int32_t human;
};

struct audio_control
{
  float   scale[4][4][GG_AUDIO_LINKS];
  float   pan  [4][4][GG_AUDIO_LINKS];
  uint8_t level[4][4][GG_AUDIO_LINKS];
  uint8_t active;
};

// ----------------------------- structs ------------------------------

struct graph_link
{
  graph_designer *designer;
  graph_node *from, *to;
  double x1, y1, x2, y2;
  int target;
  bool flipped;
  graph_arrow *arrow;
  link_mouse_t on_mouse_down, on_mouse_move, on_mouse_up;
};

struct graph_arrow
{
  graph_link *link;
  double x1, y1, x2, y2, x3, y3;
  arrow_mouse_t on_mouse_down, on_mouse_move, on_mouse_up;
};

struct graph_cord
{
  double x1, y1, x2, y2;
};

struct graph_node
{
  graph_designer *designer;
  std::vector< graph_link * > link1, link2;
  bool fixed;
  double x;
  double y;
  double dragx;
  double dragy;
  double targetx;
  double targety;
  node_mouse_t on_mouse_down, on_mouse_move, on_mouse_up;
};

struct graph_designer
```

```
{
  int id;
  graph_node *start_node;
  graph_node *end_node;
  graph_node *dragging_node;
  graph_node *cording_node;
  graph_cord *cord;
  std::vector< graph_link  * >  links;
  std::vector< graph_arrow * > arrows;
  std::vector< graph_node  * >  nodes;
  designer_mouse_t on_mouse_down, on_mouse_move, on_mouse_up;
};

struct graph_draw
{
  int components;
  float tri[4096];
  GLuint vbo_tri;
  GLuint vao_tri;
  GLuint vao_quad;
  GLuint prog_circle;
  GLuint prog_arrow;
  GLuint prog_line;
};

// ----------------------------- link ---------------------------------

graph_link *link_new(graph_designer *designer, graph_node *from, graph_node *to)
{
  graph_link *link = new graph_link();
  link->designer = designer;
  link->from = from;
  link->to = to;
  link->x1 = from->x;
  link->y1 = from->y;
  link->x2 = to->x;
  link->y2 = to->y;
  link->target = designer->id;
  link->arrow = new graph_arrow();
  link->arrow->link = link;
  from->link1.push_back(link);
  to->link2.push_back(link);
  designer->links.push_back(link);
  designer->arrows.push_back(link->arrow);
  link_update(link);
  return link;
}

void link_delete(graph_link *link)
{
  for (int i = 0; i < link->from->link1.size(); )
    if (link == link->from->link1[i])
      link->from->link1.erase(link->from->link1.begin() + i);
    else
      ++i;
  for (int i = 0; i < link->to->link2.size(); )
    if (link == link->to->link2[i])
```

```
        link->to->link2.erase(link->to->link2.begin() + i);
      else
        ++i;
    for (int i = 0; i < link->designer->links.size(); )
      if (link == link->designer->links[i])
        link->designer->links.erase(link->designer->links.begin() + i);
      else
        ++i;
    for (int i = 0; i < link->designer->arrows.size(); )
      if (link->arrow == link->designer->arrows[i])
        link->designer->arrows.erase(link->designer->arrows.begin() + i);
      else
        ++i;
    arrow_delete(link->arrow);
    link->arrow = 0;
    link->from = 0;
    link->to = 0;
    link->designer = 0;
    delete link;
  }

  void link_leave_mode(graph_link *link)
  {
    (void) link;
  }

  void link_enter_mode(graph_link *link, graph_mode mode)
  {
    (void) link;
    (void) mode;
  }

  void link_update(graph_link *link)
  {
    const double w = 30;
    const double c = w * -0.5;
    const double s = w * 0.8660254037844387;
    double x1 = link->x1;
    double y1 = link->y1;
    double x2 = link->x2;
    double y2 = link->y2;
    double dx = x2 - x1;
    double dy = y2 - y1;
    if (link->flipped)
    {
      dx = -dx;
      dy = -dy;
    }
    double r = sqrt(dx * dx + dy * dy);
    dx /= r;
    dy /= r;
    double ox = (x1 + x2) / 2;
    double oy = (y1 + y2) / 2;
    link->arrow->x1 = ox + w * dx;
    link->arrow->y1 = oy + w * dy;
    link->arrow->x2 = ox + c * dx + s * dy;
    link->arrow->y2 = oy - s * dx + c * dy;
```

```
295     link->arrow->x3 = ox + c * dx - s * dy;
        link->arrow->y3 = oy + s * dx + c * dy;
    }

    // ----------------------------- arrow --------------------------------
300
    graph_arrow *arrow_new()
    {
      graph_arrow *arrow = new graph_arrow();
      arrow->on_mouse_down = 0;
305   arrow->on_mouse_move = 0;
      arrow->on_mouse_up   = 0;
      return arrow;
    }

310 void arrow_delete(graph_arrow *arrow)
    {
      delete arrow;
    }

315 void arrow_leave_mode(graph_arrow *arrow)
    {
      arrow->on_mouse_down = 0;
      arrow->on_mouse_move = 0;
      arrow->on_mouse_up   = 0;
320 }

    bool arrow_hit(graph_arrow *arrow, double x, double y)
    {
      double x0 = (arrow->x1 + arrow->x2 + arrow->x3) / 3;
325   double y0 = (arrow->y1 + arrow->y2 + arrow->y3) / 3;
      double dx = x - x0;
      double dy = y - y0;
      double r2 = dx * dx + dy * dy;
      return r2 < 500;
330 }

    bool arrow_mouse_down_delete(graph_arrow *arrow, double x, double y)
    {
      if (arrow_hit(arrow, x, y))
335   {
        link_delete(arrow->link);
        // arrow is deleted
        return true;
      }
340   return false;
    }

    bool arrow_mouse_down_flip(graph_arrow *arrow, double x, double y)
    {
345   if (arrow_hit(arrow, x, y))
      {
        arrow->link->flipped = !(arrow->link->flipped);
        link_update(arrow->link);
        return true;
350   }
      return false;
```

```
}

bool arrow_mouse_down_target(graph_arrow *arrow, double x, double y)
{
  if (arrow_hit(arrow, x, y))
  {
    arrow->link->target = (arrow->link->target + 1) % 4;
    link_update(arrow->link);
    return true;
  }
  return false;
}

void arrow_enter_mode(graph_arrow *arrow, graph_mode mode)
{
  switch (mode)
  {
    case mode_delete_link:
      arrow->on_mouse_down = arrow_mouse_down_delete;
      break;
    case mode_flip_link:
      arrow->on_mouse_down = arrow_mouse_down_flip;
      break;
    case mode_target_link:
      arrow->on_mouse_down = arrow_mouse_down_target;
      break;
    case mode_create_link:
    case mode_create_node:
    case mode_delete_node:
    case mode_move_node:
    case mode_help:
      break;
  }
}

// ----------------------------- cord --------------------------------

graph_cord *cord_new(double x1, double y1, double x2, double y2)
{
  graph_cord *cord = new graph_cord();
  cord->x1 = x1;
  cord->y1 = y1;
  cord->x2 = x2;
  cord->y2 = y2;
  return cord;
}

void cord_delete(graph_cord *cord)
{
  delete cord;
}

// ----------------------------- node --------------------------------

graph_node *node_new(graph_designer *designer, double x, double y, bool fixed)
{
  graph_node *node = new graph_node();
```

```
  node->fixed = fixed;
  node->x = fmin(fmax(x, minimum_x), maximum_x);
  node->y = fmin(fmax(y, minimum_y), maximum_y);
  node->dragx = node->x;
  node->dragy = node->y;
  node->targetx = node->x;
  node->targety = node->y;
  node->designer = designer;
  designer->nodes.push_back(node);
  return node;
}

void node_delete(graph_node *node)
{
  graph_designer *designer = node->designer;
  for (int i = 0; i < designer->nodes.size(); )
    if (node == designer->nodes[i])
      designer->nodes.erase(designer->nodes.begin() + i);
    else
      ++i;
  while (node->link1.size())
    link_delete(node->link1[0]);
  while (node->link2.size())
    link_delete(node->link2[0]);
  node->designer = 0;
  delete node;
}

bool nodes_linked(graph_node *a, graph_node *b)
{
  for (int i = 0; i < a->link1.size(); ++i)
    if (a->link1[i]->to == b)
      return true;
  for (int i = 0; i < b->link1.size(); ++i)
    if (b->link1[i]->to == a)
      return true;
  return false;
}

void node_leave_mode(graph_node *node)
{
  node->on_mouse_down = 0;
  node->on_mouse_move = 0;
  node->on_mouse_up   = 0;
}

bool node_mouse_hit(graph_node *node, double x, double y)
{
  double dx = x - node->x;
  double dy = y - node->y;
  double r2 = dx * dx + dy * dy;
  return r2 < 1000;
}

bool node_mouse_down_new(graph_node *node, double x, double y)
{
  return node_mouse_hit(node, x, y);
```

```
}

bool node_mouse_down_delete(graph_node *node, double x, double y)
{
  if (node_mouse_hit(node, x, y))
  {
    node_delete(node);
    return true;
  }
  return false;
}

bool node_mouse_down_move(graph_node *node, double x, double y)
{
  if (node_mouse_hit(node, x, y))
  {
    node->designer->dragging_node = node;
    node->designer->on_mouse_move = designer_mouse_move_dragging;
    node->designer->on_mouse_up   = designer_mouse_move_dragged;
    node->dragx = fmin(fmax(x, minimum_x), maximum_x);
    node->dragy = fmin(fmax(y, minimum_y), maximum_y);
    return true;
  }
  return false;
}

bool node_mouse_down_link(graph_node *node, double x, double y)
{
  if (node_mouse_hit(node, x, y) && node->designer->links.size() < 8)
  {
    node->designer->cording_node = node;
    node->designer->cord = cord_new(node->x, node->y, x, y);
    for (int i = 0; i < node->designer->nodes.size(); ++i)
      if (node != node->designer->nodes[i] && ! nodes_linked(node, node->designer->nodes[i]))
        node->designer->nodes[i]->on_mouse_up = node_mouse_up_corded;
    node->designer->on_mouse_move = designer_mouse_move_cording;
    node->designer->on_mouse_up   = designer_mouse_move_corded;
    return true;
  }
  return false;
}

bool node_mouse_up_corded(graph_node *node, double x, double y)
{
  if (node_mouse_hit(node, x, y))
  {
    link_new(node->designer, node->designer->cording_node, node);
    node->designer->cording_node = 0;
    delete node->designer->cord;
    node->designer->cord = 0;
    for (int i = 0; i < node->designer->nodes.size(); ++i)
      node->designer->nodes[i]->on_mouse_up = 0;
    node->designer->on_mouse_move = 0;
    node->designer->on_mouse_up   = 0;
    return true;
  }
```

```
  return false;
}

void node_enter_mode(graph_node *node, graph_mode mode)
{
  switch (mode)
  {
    case mode_create_node:
      node->on_mouse_down = node_mouse_down_new;
      break;
    case mode_delete_node:
      if (! node->fixed)
        node->on_mouse_down = node_mouse_down_delete;
      break;
    case mode_move_node:
      if (! node->fixed)
        node->on_mouse_down = node_mouse_down_move;
      break;
    case mode_create_link:
      node->on_mouse_down = node_mouse_down_link;
      break;
    case mode_delete_link:
    case mode_flip_link:
    case mode_target_link:
    case mode_help:
      break;
  }
}

// ----------------------------- designer -----------------------------

graph_designer *designer_new(int id)
{
  graph_designer *designer = new graph_designer();
  designer->id = id;
  designer->start_node = node_new(designer, 100, 400, true);
  designer->end_node   = node_new(designer, 700, 400, true);
  designer->dragging_node = 0;
  designer->cording_node = 0;
  designer->on_mouse_down = 0;
  designer->on_mouse_move = 0;
  designer->on_mouse_up   = 0;
  return designer;
}

void designer_leave_mode(graph_designer *designer)
{
  for (int i = 0; i < designer->links.size(); ++i)
    link_leave_mode(designer->links[i]);
  for (int i = 0; i < designer->arrows.size(); ++i)
    arrow_leave_mode(designer->arrows[i]);
  for (int i = 0; i < designer->nodes.size(); ++i)
    node_leave_mode(designer->nodes[i]);
  designer->on_mouse_down = 0;
  designer->on_mouse_move = 0;
  designer->on_mouse_up   = 0;
}
```

```
580 void designer_enter_mode(graph_designer *designer, graph_mode mode)
    {
      for (int i = 0; i < designer->links.size(); ++i)
        link_enter_mode(designer->links[i], mode);
      for (int i = 0; i < designer->arrows.size(); ++i)
585     arrow_enter_mode(designer->arrows[i], mode);
      for (int i = 0; i < designer->nodes.size(); ++i)
        node_enter_mode(designer->nodes[i], mode);
      switch (mode)
      {
590     case mode_create_node:
          designer->on_mouse_down = designer_mouse_down_node_new;
          break;
        case mode_delete_node:
        case mode_move_node:
595     case mode_create_link:
        case mode_delete_link:
        case mode_flip_link:
        case mode_target_link:
        case mode_help:
600       break;
      }
    }

    void node_move_by(graph_node *node, double dx, double dy)
605 {
      for (int i = 0; i < node->link2.size(); ++i)
      {
        node->link2[i]->x2 = fmin(fmax(node->link2[i]->x2 + dx, minimum_x), ↙
            ↳ maximum_x);
        node->link2[i]->y2 = fmin(fmax(node->link2[i]->y2 + dy, minimum_y), ↙
            ↳ maximum_y);
610     link_update(node->link2[i]);
      }
      for (int i = 0; i < node->link1.size(); ++i)
      {
        node->link1[i]->x1 = fmin(fmax(node->link1[i]->x1 + dx, minimum_x), ↙
            ↳ maximum_x);
615     node->link1[i]->y1 = fmin(fmax(node->link1[i]->y1 + dy, minimum_y), ↙
            ↳ maximum_y);
        link_update(node->link1[i]);
      }
      node->x = fmin(fmax(node->x + dx, minimum_x), maximum_x);
      node->y = fmin(fmax(node->y + dy, minimum_y), maximum_y);
620 }

    bool designer_mouse_down(graph_designer *designer, double x, double y)
    {
      for (int i = 0; i < designer->nodes.size(); ++i)
625     if (designer->nodes[i]->on_mouse_down)
          if (designer->nodes[i]->on_mouse_down(designer->nodes[i], x, y))
            return true;
      for (int i = 0; i < designer->arrows.size(); ++i)
        if (designer->arrows[i]->on_mouse_down)
630       if (designer->arrows[i]->on_mouse_down(designer->arrows[i], x, y))
            return true;
```

```
  if (designer->on_mouse_down)
    return designer->on_mouse_down(designer, x, y);
  return false;
}

bool designer_mouse_move(graph_designer *designer, double x, double y)
{
  for (int i = 0; i < designer->nodes.size(); ++i)
    if (designer->nodes[i]->on_mouse_move)
      if (designer->nodes[i]->on_mouse_move(designer->nodes[i], x, y))
        return true;
  if (designer->on_mouse_move)
    return designer->on_mouse_move(designer, x, y);
  return false;
}

bool designer_mouse_up(graph_designer *designer, double x, double y)
{
  for (int i = 0; i < designer->nodes.size(); ++i)
    if (designer->nodes[i]->on_mouse_up)
      if (designer->nodes[i]->on_mouse_up(designer->nodes[i], x, y))
        return true;
  if (designer->on_mouse_up)
    return designer->on_mouse_up(designer, x, y);
  return false;
}

bool designer_mouse_down_node_new(graph_designer *designer, double x, double y)
{
  if (minimum_x <= x && x <= maximum_x && minimum_y <= y && y <= maximum_y &&
      ↳ designer->nodes.size() < 8)
  {
    graph_node *node = node_new(designer, x, y, false);
    node_enter_mode(node, mode_create_node);
  }
  return true;
}

bool designer_mouse_move_dragging(graph_designer *designer, double x, double y)
{
  double dx = fmin(fmax(x, minimum_x), maximum_x) - designer->dragging_node->
      ↳ dragx;
  double dy = fmin(fmax(y, minimum_y), maximum_y) - designer->dragging_node->
      ↳ dragy;
  designer->dragging_node->dragx = fmin(fmax(x, minimum_x), maximum_x);
  designer->dragging_node->dragy = fmin(fmax(y, minimum_y), maximum_y);
  node_move_by(designer->dragging_node, dx, dy);
  return true;
}

bool designer_mouse_move_dragged(graph_designer *designer, double x, double y)
{
  designer_mouse_move_dragging(designer, x, y);
  designer->on_mouse_move = 0;
  designer->on_mouse_up = 0;
  designer->dragging_node = 0;
  return true;
```

```
}

bool designer_mouse_move_cording(graph_designer *designer, double x, double y)
{
  designer->cord->x2 = fmin(fmax(x, minimum_x), maximum_x);
  designer->cord->y2 = fmin(fmax(y, minimum_y), maximum_y);
  return true;
}

bool designer_mouse_move_corded(graph_designer *designer, double x, double y)
{
  for (int i = 0; i < designer->nodes.size(); ++i)
    designer->nodes[i]->on_mouse_up = 0;
  designer->on_mouse_move = 0;
  designer->on_mouse_up = 0;
  designer->cording_node = 0;
  delete designer->cord;
  designer->cord = 0;
  return true;
  (void) x;
  (void) y;
}

// ------------------------------ drawing ------------------------------

int draw_enqueue_triangle(graph_draw *draw, int k, double x1, double y1, double x2, double y2, double x3, double y3, glm::vec3 colour)
{
  draw->tri[k++] = x1;
  draw->tri[k++] = y1;
  draw->tri[k++] = 1;
  draw->tri[k++] = 0;
  draw->tri[k++] = 0;
  draw->tri[k++] = colour[0];
  draw->tri[k++] = colour[1];
  draw->tri[k++] = colour[2];
  draw->tri[k++] = x2;
  draw->tri[k++] = y2;
  draw->tri[k++] = 0;
  draw->tri[k++] = 1;
  draw->tri[k++] = 0;
  draw->tri[k++] = colour[0];
  draw->tri[k++] = colour[1];
  draw->tri[k++] = colour[2];
  draw->tri[k++] = x3;
  draw->tri[k++] = y3;
  draw->tri[k++] = 0;
  draw->tri[k++] = 0;
  draw->tri[k++] = 1;
  draw->tri[k++] = colour[0];
  draw->tri[k++] = colour[1];
  draw->tri[k++] = colour[2];
  return k;
}

int draw_enqueue_quad(graph_draw *draw, int k, double x1, double y1, double x2,
```

```
    ↪ double y2, double x3, double y3, double x4, double y4, glm::vec3 colour)
{
  draw->tri[k++] = x1;
  draw->tri[k++] = y1;
  draw->tri[k++] = 0;
  draw->tri[k++] = 0;
  draw->tri[k++] = colour[0];
  draw->tri[k++] = colour[1];
  draw->tri[k++] = colour[2];
  draw->tri[k++] = x2;
  draw->tri[k++] = y2;
  draw->tri[k++] = 0;
  draw->tri[k++] = 1;
  draw->tri[k++] = colour[0];
  draw->tri[k++] = colour[1];
  draw->tri[k++] = colour[2];
  draw->tri[k++] = x3;
  draw->tri[k++] = y3;
  draw->tri[k++] = 1;
  draw->tri[k++] = 0;
  draw->tri[k++] = colour[0];
  draw->tri[k++] = colour[1];
  draw->tri[k++] = colour[2];
  draw->tri[k++] = x3;
  draw->tri[k++] = y3;
  draw->tri[k++] = 1;
  draw->tri[k++] = 0;
  draw->tri[k++] = colour[0];
  draw->tri[k++] = colour[1];
  draw->tri[k++] = colour[2];
  draw->tri[k++] = x2;
  draw->tri[k++] = y2;
  draw->tri[k++] = 0;
  draw->tri[k++] = 1;
  draw->tri[k++] = colour[0];
  draw->tri[k++] = colour[1];
  draw->tri[k++] = colour[2];
  draw->tri[k++] = x4;
  draw->tri[k++] = y4;
  draw->tri[k++] = 1;
  draw->tri[k++] = 1;
  draw->tri[k++] = colour[0];
  draw->tri[k++] = colour[1];
  draw->tri[k++] = colour[2];
  return k;
}

int draw_enqueue_line(graph_draw *draw, int k, double x1, double y1, double x2, ↙
    ↪ double y2, double w, glm::vec3 colour)
{
  double dx = x2 - x1;
  double dy = y2 - y1;
  double r = sqrt(dx * dx + dy * dy);
  dx /= r;
  dy /= r;
  k = draw_enqueue_quad(draw, k, x1 - w * dy, y1 + w * dx, x2 - w * dy, y2 + w *↙
```

```
      ↪ dx, x1 + w * dy, y1 - w * dx, x2 + w * dy, y2 - w * dx, colour);
  return k;
}

int draw_enqueue_circle(graph_draw *draw, int k, double x, double y, double w, ↙
    ↪ glm::vec3 colour)
{
  k = draw_enqueue_quad(draw, k, x - w, y - w, x + w, y - w, x - w, y + w, x + w↙
      ↪ , y + w, colour);
  return k;
}

void draw_designer(graph_draw *draw, graph_designer *designer, bool is_active, ↙
    ↪ bool big)
{
  glm::vec3 bg = glm::mix(colours[designer->id], white, 0.75f);
  if (! is_active)
    bg = glm::mix(bg, black, 0.5f);
  glClearColor(bg[0], bg[1], bg[2], 0);
  glClear(GL_COLOR_BUFFER_BIT);
  int k = 0;
  for (int i = 0; i < designer->links.size() && k <= draw->components - 7 * 6; ↙
      ↪ ++i)
  {
    double w = big ? 8 : 16;
    double x1 = designer->links[i]->x1;
    double y1 = designer->links[i]->y1;
    double x2 = designer->links[i]->x2;
    double y2 = designer->links[i]->y2;
    glm::vec3 colour = colours[designer->links[i]->target];
    k = draw_enqueue_line(draw, k, x1, y1, x2, y2, w, colour);
  }
  if (designer->cord)
  {
    double w = big ? 8 : 16;
    double x1 = designer->cord->x1;
    double y1 = designer->cord->y1;
    double x2 = designer->cord->x2;
    double y2 = designer->cord->y2;
    k = draw_enqueue_line(draw, k, x1, y1, x2, y2, w, colours[designer->id]);
  }
  glBufferSubData(GL_ARRAY_BUFFER, 0, k * sizeof(float), &draw->tri[0]);
  glBindVertexArray(draw->vao_quad);
  glUseProgram(draw->prog_line);
  glDrawArrays(GL_TRIANGLES, 0, k / 7);
  k = 0;
  for (int i = 0; i < designer->arrows.size() && k < draw->components - 8 * 3; ↙
      ↪ ++i)
  {
    double x1 = designer->arrows[i]->x1;
    double y1 = designer->arrows[i]->y1;
    double x2 = designer->arrows[i]->x2;
    double y2 = designer->arrows[i]->y2;
    double x3 = designer->arrows[i]->x3;
    double y3 = designer->arrows[i]->y3;
    glm::vec3 colour = colours[designer->arrows[i]->link->target];
    k = draw_enqueue_triangle(draw, k, x1, y1, x2, y2, x3, y3, colour);
```

```
    }
    glBufferSubData(GL_ARRAY_BUFFER, 0, k * sizeof(float), &draw->tri[0]);
    glBindVertexArray(draw->vao_tri);
    glUseProgram(draw->prog_arrow);
    glDrawArrays(GL_TRIANGLES, 0, k / 8);
    k = 0;
    for (int i = 0; i < designer->nodes.size() && k <= draw->components - 7 * 6; ↙
        ↳ ++i)
    {
      const double w = 20;
      double x = designer->nodes[i]->x;
      double y = designer->nodes[i]->y;
      glm::vec3 colour = colours[designer->nodes[i]->designer->id];
      if (designer->nodes[i]->fixed)
        colour = white;
      k = draw_enqueue_circle(draw, k, x, y, w, colour);
    }
    glBufferSubData(GL_ARRAY_BUFFER, 0, k * sizeof(float), &draw->tri[0]);
    glBindVertexArray(draw->vao_quad);
    glUseProgram(draw->prog_circle);
    glDrawArrays(GL_TRIANGLES, 0, k / 7);
  }

  void debug_program(GLuint program) {
    GLint status = 0;
    glGetProgramiv(program, GL_LINK_STATUS, &status);
    GLint length = 0;
    glGetProgramiv(program, GL_INFO_LOG_LENGTH, &length);
    char *info = 0;
    if (length) {
      info = (char *)malloc(length + 1);
      info[0] = 0;
      glGetProgramInfoLog(program, length, 0, info);
      info[length] = 0;
    }
    if ((info && info[0]) || ! status) {
      printf("program link info:\n%s", info ? info : "(no info log)");
    }
    if (info) {
      free(info);
    }
  }

  void debug_shader(GLuint shader, GLenum type, const char *source) {
    GLint status = 0;
    glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
    GLint length = 0;
    glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &length);
    char *info = 0;
    if (length) {
      info = (char *)malloc(length + 1);
      info[0] = 0;
      glGetShaderInfoLog(shader, length, 0, info);
      info[length] = 0;
    }
    if ((info && info[0]) || ! status) {
      const char *type_str = "unknown";
```

```
    switch (type) {
      case GL_VERTEX_SHADER: type_str = "vertex"; break;
      case GL_FRAGMENT_SHADER: type_str = "fragment"; break;
    }
    printf("%s shader compile info:\n%s\nshader source:\n%s", type_str, info ? info : "(no info log)", source ? source : "(no source)");
  }
  if (info) {
    free(info);
  }
}

GLuint vertex_fragment_shader(const char *vert, const char *frag) {
  GLuint program = glCreateProgram();
  {
    GLuint shader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(shader, 1, &vert, 0);
    glCompileShader(shader);
    debug_shader(shader, GL_VERTEX_SHADER, vert);
    glAttachShader(program, shader);
    glDeleteShader(shader);
  }
  {
    GLuint shader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(shader, 1, &frag, 0);
    glCompileShader(shader);
    debug_shader(shader, GL_FRAGMENT_SHADER, frag);
    glAttachShader(program, shader);
    glDeleteShader(shader);
  }
  glBindAttribLocation(program, 0, "pos");
  glBindAttribLocation(program, 1, "tcd");
  glBindAttribLocation(program, 2, "clr");
  glLinkProgram(program);
  debug_program(program);
  return program;
}

const char *draw_circle_vert =
"#version 100\n"
"precision highp float;\n"
"attribute vec2 pos;\n"
"attribute vec2 tcd;\n"
"attribute vec3 clr;\n"
"varying vec2 v_texcoord;\n"
"varying vec3 v_colour;\n"
"void main() {\n"
"  gl_Position = vec4(2.0/800.0 * pos - vec2(1.0,1.0), 0.0, 1.0);\n"
"  v_texcoord = tcd * 2.0 - vec2(1.0, 1.0);\n"
"  v_colour = clr;\n"
"}\n"
;

const char *draw_circle_frag =
"#version 100\n"
"#extension GL_OES_standard_derivatives : require\n"
"precision highp float;\n"
```

```
960   "varying vec2 v_texcoord;\n"
      "varying vec3 v_colour;\n"
      "void main() {\n"
      "  float r = length(v_texcoord);\n"
      "  float d = mix(length(dFdx(v_texcoord)), length(dFdy(v_texcoord)), 0.5);\n"
965   "  gl_FragColor = vec4(v_colour * (smoothstep(0.2, 0.2 + d, r) - smoothstep(0.8 ↙
          ↳ - d, 0.8, r)), 1.0 - smoothstep(1.0 - d, 1.0, r));\n"
      "}\n"
      ;

      const char *draw_arrow_vert =
970   "#version 100\n"
      "precision highp float;\n"
      "attribute vec2 pos;\n"
      "attribute vec3 tcd;\n"
      "attribute vec3 clr;\n"
975   "varying vec3 v_texcoord;\n"
      "varying vec3 v_colour;\n"
      "void main() {\n"
      "  gl_Position = vec4(2.0/800.0 * pos - vec2(1.0,1.0), 0.0, 1.0);\n"
      "  v_texcoord = tcd;\n"
980   "  v_colour = clr;\n"
      "}\n"
      ;

      const char *draw_arrow_frag =
985   "#version 100\n"
      "#extension GL_OES_standard_derivatives : require\n"
      "precision highp float;\n"
      "varying vec3 v_texcoord;\n"
      "varying vec3 v_colour;\n"
990   "void main() {\n"
      "  vec3 w = fwidth(v_texcoord);\n"
      "  vec3 a = smoothstep(vec3(0.1), vec3(0.1) + w, v_texcoord);\n"
      "  vec3 b = smoothstep(vec3(0.0), vec3(0.0) + w, v_texcoord);\n"
      "  float e1 = max(min(min(a.x, a.y), a.z), 1.0 - smoothstep(0.05, 0.05 + length(↙
          ↳ w), abs(v_texcoord.y - v_texcoord.z)));\n"
995   "  float e2 = min(min(b.x, b.y), b.z);\n"
      "  gl_FragColor = vec4(v_colour * e1, e2);\n"
      "}\n"
      ;

1000  const char *draw_line_vert =
      "#version 100\n"
      "precision highp float;\n"
      "attribute vec2 pos;\n"
      "attribute vec2 tcd;\n"
1005  "attribute vec3 clr;\n"
      "varying vec2 v_texcoord;\n"
      "varying vec3 v_colour;\n"
      "void main() {\n"
      "  gl_Position = vec4(2.0/800.0 * pos - vec2(1.0,1.0), 0.0, 1.0);\n"
1010  "  v_texcoord = tcd * 2.0 - vec2(1.0, 1.0);\n"
      "  v_colour = clr;\n"
      "}\n"
      ;
```

```
1015  const char *draw_line_frag =
      "#version 100\n"
      "#extension GL_OES_standard_derivatives : require\n"
      "precision highp float;\n"
      "varying vec2 v_texcoord;\n"
1020  "varying vec3 v_colour;\n"
      "void main() {\n"
      "  float r = abs(v_texcoord.x);\n"
      "  float d = abs(dFdx(v_texcoord.x)) + abs(dFdy(v_texcoord.x));\n"
      "  gl_FragColor = vec4(v_colour * (1.0 - smoothstep(0.333 - d, 0.333, r)), 1.0 -↙
          ↳ smoothstep(1.0 - d, 1.0, r));\n"
1025  "}\n"
      ;

      graph_draw *draw_new()
      {
1030    graph_draw *draw = new graph_draw();
        draw->components = 4096;
        glGenBuffers(1, &draw->vbo_tri);
        glBindBuffer(GL_ARRAY_BUFFER, draw->vbo_tri);
        glBufferData(GL_ARRAY_BUFFER, draw->components * sizeof(GLfloat), 0, ↙
            ↳ GL_DYNAMIC_DRAW);
1035    glGenVertexArrays(1, &draw->vao_tri);
        glBindVertexArray(draw->vao_tri);
        glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), 0);
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), ((char *)↙
            ↳ 0) + 2 * sizeof(GLfloat));
        glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), ((char *)↙
            ↳ 0) + 5 * sizeof(GLfloat));
1040    glEnableVertexAttribArray(0);
        glEnableVertexAttribArray(1);
        glEnableVertexAttribArray(2);
        glGenVertexArrays(1, &draw->vao_quad);
        glBindVertexArray(draw->vao_quad);
1045    glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 7 * sizeof(GLfloat), 0);
        glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 7 * sizeof(GLfloat), ((char *)↙
            ↳ 0) + 2 * sizeof(GLfloat));
        glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 7 * sizeof(GLfloat), ((char *)↙
            ↳ 0) + 4 * sizeof(GLfloat));
        glEnableVertexAttribArray(0);
        glEnableVertexAttribArray(1);
1050    glEnableVertexAttribArray(2);
        draw->prog_circle  = vertex_fragment_shader(draw_circle_vert,  ↙
            ↳ draw_circle_frag);
        draw->prog_arrow   = vertex_fragment_shader(draw_arrow_vert,   draw_arrow_frag↙
            ↳ );
        draw->prog_line    = vertex_fragment_shader(draw_line_vert,    draw_line_frag)↙
            ↳ ;
        glDisable(GL_DEPTH_TEST);
1055    glEnable(GL_SCISSOR_TEST);
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        return draw;
      }
1060
      struct {
        graph_mode current_mode;
```

```
  int active;
  graph_designer *designer[4];
  graph_draw *draw;
  double x, y;
  pthread_t thread;
  pthread_mutex_t mutex;
  audio_control audio;
  video_control video;
  bool avdata;
  bool quit;
  time_t human;
} S;

void motioncb(double x, double y)
{
  y = screen_height - 1 - y;
  const double dx = (screen_width - screen_height) * 0.5;
  const double f = 800.0 / screen_height;
  designer_mouse_move(S.designer[S.active], (x - dx) * f, y * f);
  S.x = x;
  S.y = y;
}

void buttoncb(int action)
{
  const double button_size = 0.5 * (screen_width - screen_height) - 2 * screen_border;
  int active = -1;
  S.human = time(0);
  if (screen_border <= S.x && S.x < screen_border + button_size)
  {
    if (0.5 * (screen_height - screen_border) - button_size <= S.y && S.y < 0.5 * (screen_height - screen_border)) active = 1;
    if (0.5 * (screen_height + screen_border) <= S.y && S.y < 0.5 * (screen_height + screen_border) + button_size) active = 0;
  }
  if (screen_width - screen_border - button_size <= S.x && S.x < screen_width - screen_border)
  {
    if (0.5 * (screen_height - screen_border) - button_size <= S.y && S.y < 0.5 * (screen_height - screen_border)) active = 2;
    if (0.5 * (screen_height + screen_border) <= S.y && S.y < 0.5 * (screen_height + screen_border) + button_size) active = 3;
  }
  if (active != -1)
  {
    designer_leave_mode(S.designer[S.active]);
    S.active = active;
    designer_enter_mode(S.designer[S.active], S.current_mode);
  }
  else
  {
    const double dx = 0.5 * (screen_width - screen_height);
    const double f = 800.0 / screen_height;
    double x = S.x - dx;
    if (action == 1)
      designer_mouse_down(S.designer[S.active], x * f, S.y * f);
```

```
        if (action == -1)
1115      designer_mouse_up(S.designer[S.active], x * f, S.y * f);
    }
}

static bool onTouch(GLFMDisplay *display, int touch, GLFMTouchPhase phase, ↙
    ↪ double x, double y) {
1120      motioncb(x, y);
          switch (phase)
          {
                  case GLFMTouchPhaseHover:
                      return false;
1125              case GLFMTouchPhaseBegan:
                      buttoncb(1);
                      return true;
                  case GLFMTouchPhaseEnded:
                  case GLFMTouchPhaseCancelled:
1130                  buttoncb(-1);
                  return true;
                  case GLFMTouchPhaseMoved:
                      return true;
          }
1135  }

static void *networkSendThread(void *arg)
{
          (void) arg;
1140    printf("Hello from thread!\n");
#ifdef HAVE_LO
          const char *video_host = "solambgel.config";
          const char *audio_host = video_host;
          lo_address at = lo_address_new(audio_host, "6060");
1145      lo_address vt = lo_address_new(video_host, "6061");
#endif
          do
          {
                  if (0 == pthread_mutex_lock(&S.mutex))
1150              {
      time_t now = time(0);
                      bool quit = false;
                      audio_control a;
                      video_control v;
1155                  bool avdata = false;
                      quit = S.quit;
                      avdata = S.avdata;
                      if (avdata)
                      {
1160                          memcpy(&a, &S.audio, sizeof(a));
                              memcpy(&v, &S.video, sizeof(v));
                              S.avdata = false;
                      }
      v.human = now - S.human;
1165                  pthread_mutex_unlock(&S.mutex);
                      if (quit) break;
                      if (avdata)
                      {
#ifdef HAVE_LO
```

```
1170                                lo_blob ablob = lo_blob_new(sizeof(a), &a);
                                    if (ablob)
                                    {
                                      if (lo_send(at, "/audio", "b", ablob) == -1)
                                        printf("OSC error %d: %s\n", lo_address_errno(at↙
                                            ↳ ), lo_address_errstr(at));
1175                                  lo_blob_free(ablob);
                                    }
                                    else
                                      printf("OSC error: couldn't lo_blob_new\n");
                                    lo_blob vblob = lo_blob_new(sizeof(v), &v);
1180                                if (vblob)
                                    {
                                      if (lo_send(vt, "/video", "b", vblob) == -1)
                                        printf("OSC error %d: %s\n", lo_address_errno(vt↙
                                            ↳ ), lo_address_errstr(vt));
                                      lo_blob_free(vblob);
1185                                }
                                    else
                                      printf("OSC error: couldn't lo_blob_new\n");
    #endif
                            }
1190                    }
                        usleep(1000000 / 60);
            } while (1);
            return 0;
    }
1195


    static void onSurfaceCreated(GLFMDisplay *display, int width, int height) {
            screen_width = width;
1200        screen_height = height;
            screen_border = fmin(width, height) / 40;

        GLFMRenderingAPI api = glfmGetRenderingAPI(display);
        printf("Hello from GLFM! Using OpenGL %s\n",
1205            api == GLFMRenderingAPIOpenGLES32 ? "ES 3.2" :
                api == GLFMRenderingAPIOpenGLES31 ? "ES 3.1" :
                api == GLFMRenderingAPIOpenGLES3 ? "ES 3.0" : "ES 2.0");

    glGenVertexArraysOES = (PFNGLGENVERTEXARRAYSOESPROC)eglGetProcAddress ("↙
        ↳ glGenVertexArraysOES");
1210 glBindVertexArrayOES = (PFNGLBINDVERTEXARRAYOESPROC)eglGetProcAddress ("↙
        ↳ glBindVertexArrayOES");
    glDeleteVertexArraysOES = (PFNGLDELETEVERTEXARRAYSOESPROC)eglGetProcAddress ("↙
        ↳ glDeleteVertexArraysOES");
    glIsVertexArrayOES = (PFNGLISVERTEXARRAYOESPROC)eglGetProcAddress ("↙
        ↳ glIsVertexArrayOES");

      if (S.draw) delete S.draw;
1215  S.draw = draw_new();
      pthread_create(&S.thread, nullptr, networkSendThread, nullptr);
    }

    static void onSurfaceDestroyed(GLFMDisplay *display)
1220 {
```

```
  pthread_join(S.thread, nullptr);
}

static void onFrame(GLFMDisplay *display, double frameTime) {

    const double button_size = 0.5 * (screen_width - screen_height) - 2 * screen_border;
    glScissor(0, 0, screen_width, screen_height);
    glViewport(0, 0, screen_width, screen_height);
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);

    glScissor(screen_border, 0.5 * (screen_height - screen_border) - button_size, button_size, button_size);
    glViewport(screen_border, 0.5 * (screen_height - screen_border) - button_size, button_size, button_size);
    draw_designer(S.draw, S.designer[1], 1 == S.active, false);

    glScissor(screen_border, 0.5 * (screen_height + screen_border), button_size, button_size);
    glViewport(screen_border, 0.5 * (screen_height + screen_border), button_size, button_size);
    draw_designer(S.draw, S.designer[0], 0 == S.active, false);

    glScissor(screen_width - screen_border - button_size, 0.5 * (screen_height - screen_border) - button_size, button_size, button_size);
    glViewport(screen_width - screen_border - button_size, 0.5 * (screen_height - screen_border) - button_size, button_size, button_size);
    draw_designer(S.draw, S.designer[2], 2 == S.active, false);

    glScissor(screen_width - screen_border - button_size, 0.5 * (screen_height + screen_border), button_size, button_size);
    glViewport(screen_width - screen_border - button_size, 0.5 * (screen_height + screen_border), button_size, button_size);
    draw_designer(S.draw, S.designer[3], 3 == S.active, false);

    glScissor(0.5 * (screen_width - screen_height), 0, screen_height, screen_height);
    glViewport(0.5 * (screen_width - screen_height), 0, screen_height, screen_height);
    draw_designer(S.draw, S.designer[S.active], true, true);

    GLint e;
    while ((e = glGetError()))
      printf("OpenGL ERROR %d\n", e);

    audio_control a;
    memset(&a, 0, sizeof(a));
    a.active = S.active;
    for (int i = 0; i < 4; ++i)
      for (int k = 0; k < S.designer[i]->links.size(); ++k)
      {
        graph_link *l = S.designer[i]->links[k];
        int j = l->target;
        double dx = l->x2 - l->x1;
        double dy = l->y2 - l->y1;
```

```
          double ox = l->x2 + l->x1;
          // FIXME breakage may occur here, some links might be overwritten
          a.scale[i][j][k % GG_AUDIO_LINKS] = sqrt(dx * dx + dy * dy) / 600;
1270      a.pan  [i][j][k % GG_AUDIO_LINKS] = (0.5 * ox - 400) / 400 * 0.5 + 0.5;
          a.level[i][j][k % GG_AUDIO_LINKS] = 1;
        }

      video_control v;
1275  memset(&v, 0, sizeof(v));
      v.active = S.active;
      for (int i = 0; i < 4; ++i)
      {
        v.count[i] = S.designer[i]->links.size();
1280    for (int j = 0; j < S.designer[i]->links.size() && j < GG_VIDEO_LINKS; ++j↙
          ↳ )
        {
          graph_link *l = S.designer[i]->links[j];
          double dx = l->x2 - l->x1;
          double dy = l->y2 - l->y1;
1285      if (l->flipped)
          {
            dx = -dx;
            dy = -dy;
          }
1290      double ox = (l->x2 + l->x1) / 2;
          double oy = (l->y2 + l->y1) / 2;
          double s  = sqrt(dx * dx + dy * dy);
          dx /= 600;
          dy /= 600;
1295      s  /= 600;
          ox -= 400;
          oy -= 400;
          ox /= 600;
          oy /= 600;
1300      oy = -oy;
          v.source[i][j] = l->target;
          v.scale [i][j] = s;
          glm::mat3 transform = glm::mat3(float(dx), float(dy), float(ox), float(-↙
            ↳ dy), float(dx), float(oy), 0.0f, 0.0f, 1.0f);
          transform = glm::inverse(transform);
1305      for (int p = 0; p < 3; ++p)
            for (int q = 0; q < 3; ++q)
              v.transform[i][j][p][q] = transform[p][q];
        }
      }
1310
      if (0 == pthread_mutex_lock(&S.mutex))
      {
              memcpy(&S.audio, &a, sizeof(S.audio));
              memcpy(&S.video, &v, sizeof(S.video));
1315          S.avdata = true;
              pthread_mutex_unlock(&S.mutex);
        }

    time_t now = time(0);
1320  if (now >= S.human + 3 * 60)
    {
```

```
    for (int d = 0; d < 4; ++d)
    {
      for (auto node : S.designer[d]->nodes)
      {
        double dt = 0.0001;
        double dx = node->targetx - node->x;
        double dy = node->targety - node->y;
        node_move_by(node, dx * dt, dy * dt);
        if (! node->fixed)
        {
          double dz = dx * dx + dy * dy;
          if (dz < 25)
          {
            node->targetx = 50 + (750 - 50) * (rand() / (double) RAND_MAX);
            node->targety = 50 + (750 - 50) * (rand() / (double) RAND_MAX);
          }
        }
      }
    }
  }

  extern void glfmMain(GLFMDisplay *display)
  {
    memset(&S, 0, sizeof(S));
    srand(time(0));
    pthread_mutex_init(&S.mutex, nullptr);
    S.designer[0] = designer_new(0);
    S.designer[1] = designer_new(1);
    S.designer[2] = designer_new(2);
    S.designer[3] = designer_new(3);
    {
      auto nab = node_new(S.designer[0], 300, 400, false);
      auto nbc = node_new(S.designer[0], 400, 400 + 200 * sqrt(3)/2, false);
      auto ncd = node_new(S.designer[0], 500, 400, false);
      auto la = link_new(S.designer[0], S.designer[0]->start_node, nab);
      auto lb = link_new(S.designer[0], nab, nbc);
      auto lc = link_new(S.designer[0], nbc, ncd);
      auto ld = link_new(S.designer[0], ncd, S.designer[0]->end_node);
      la->target = 0;
      lb->target = 1;
      lc->target = 2;
      ld->target = 3;
      link_update(la);
      link_update(lb);
      link_update(lc);
      link_update(ld);
    }
    {
      double z = 600 / (1 + sqrt(2));
      auto nab = node_new(S.designer[1], 100 + z, 400, false);
      auto nbc = node_new(S.designer[1], 100 + z * (1 + sqrt(2)/2), 400 + z * sqrt↙
          ↳ (2)/2, false);
      auto ncd = node_new(S.designer[1], 100 + z, 400 + z * sqrt(2), false);
      auto la = link_new(S.designer[1], S.designer[1]->start_node, nab);
      auto lb = link_new(S.designer[1], nab, nbc);
      auto lc = link_new(S.designer[1], nbc, ncd);
```

```
    auto ld = link_new(S.designer[1], nbc, S.designer[1]->end_node);
    la->target = 1;
    lb->target = 2;
    lc->target = 3;
    ld->target = 0;
    link_update(la);
    link_update(lb);
    link_update(lc);
    link_update(ld);
  }
  {
    auto nab = node_new(S.designer[2], 400, 400, false);
    auto nbc = node_new(S.designer[2], 550, 400 - 300 * sqrt(3)/2, false);
    auto ncd = node_new(S.designer[2], 250, 400 - 300 * sqrt(3)/2, false);
    auto la = link_new(S.designer[2], S.designer[2]->start_node, nab);
    auto lb = link_new(S.designer[2], nab, nbc);
    auto lc = link_new(S.designer[2], nbc, ncd);
    auto ld = link_new(S.designer[2], nab, S.designer[2]->end_node);
    la->target = 3;
    lb->target = 1;
    lc->target = 2;
    ld->target = 0;
    link_update(la);
    link_update(lb);
    link_update(lc);
    link_update(ld);
  }
  {
    auto nab = node_new(S.designer[3], 400, 400, false);
    auto nbc = node_new(S.designer[3], 400, 700, false);
    auto ncd = node_new(S.designer[3], 400, 100, false);
    auto la = link_new(S.designer[3], S.designer[3]->start_node, nab);
    auto lb = link_new(S.designer[3], nab, nbc);
    auto lc = link_new(S.designer[3], nab, ncd);
    auto ld = link_new(S.designer[3], nab, S.designer[3]->end_node);
    la->target = 0;
    lb->target = 3;
    lc->target = 2;
    ld->target = 1;
    link_update(la);
    link_update(lb);
    link_update(lc);
    link_update(ld);
  }
  designer_enter_mode(S.designer[S.active = 0], S.current_mode = mode_move_node)↙
    ↳ ;

  glfmSetDisplayConfig(display,
                       GLFMRenderingAPIOpenGLES2,
                       GLFMColorFormatRGBA8888,
                       GLFMDepthFormatNone,
                       GLFMStencilFormat8,
                       GLFMMultisample4X);
  glfmSetUserData(display, &S);
      glfmSetDisplayChrome(display, GLFMUserInterfaceChromeFullscreen);
      glfmSetSurfaceCreatedFunc(display, onSurfaceCreated);
  glfmSetSurfaceResizedFunc(display, onSurfaceCreated);
```

```
    glfmSetSurfaceDestroyedFunc(display, onSurfaceDestroyed);
    glfmSetMainLoopFunc(display, onFrame);
    glfmSetTouchFunc(display, onTouch);

}
```

36 example/src/main.c

```
// Example app that draws a triangle. The triangle can be moved via touch or keyboard arrow keys.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "glfm.h"
#define FILE_COMPAT_ANDROID_ACTIVITY glfmAndroidGetActivity()
#include "file_compat.h"

typedef struct {
    GLuint program;
    GLuint vertexBuffer;

    double lastTouchX;
    double lastTouchY;

    double offsetX;
    double offsetY;
} ExampleApp;

static void onFrame(GLFMDisplay *display, double frameTime);
static void onSurfaceCreated(GLFMDisplay *display, int width, int height);
static void onSurfaceDestroyed(GLFMDisplay *display);
static bool onTouch(GLFMDisplay *display, int touch, GLFMTouchPhase phase, double x, double y);
static bool onKey(GLFMDisplay *display, GLFMKey keyCode, GLFMKeyAction action, int modifiers);

// Main entry point
void glfmMain(GLFMDisplay *display) {
    ExampleApp *app = calloc(1, sizeof(ExampleApp));

    glfmSetDisplayConfig(display,
                         GLFMRenderingAPIOpenGLES2,
                         GLFMColorFormatRGBA8888,
                         GLFMDepthFormatNone,
                         GLFMStencilFormatNone,
                         GLFMMultisampleNone);
    glfmSetUserData(display, app);
    glfmSetSurfaceCreatedFunc(display, onSurfaceCreated);
    glfmSetSurfaceResizedFunc(display, onSurfaceCreated);
    glfmSetSurfaceDestroyedFunc(display, onSurfaceDestroyed);
    glfmSetMainLoopFunc(display, onFrame);
    glfmSetTouchFunc(display, onTouch);
    glfmSetKeyFunc(display, onKey);
}

static bool onTouch(GLFMDisplay *display, int touch, GLFMTouchPhase phase, double x, double y) {
```

```
    if (phase == GLFMTouchPhaseHover) {
        return false;
    }
    ExampleApp *app = glfmGetUserData(display);
    if (phase != GLFMTouchPhaseBegan) {
        int width, height;
        glfmGetDisplaySize(display, &width, &height);
        app->offsetX += 2 * (x - app->lastTouchX) / width;
        app->offsetY -= 2 * (y - app->lastTouchY) / height;
    }
    app->lastTouchX = x;
    app->lastTouchY = y;
    return true;
}

static bool onKey(GLFMDisplay *display, GLFMKey keyCode, GLFMKeyAction action, int modifiers) {
    bool handled = false;
    if (action == GLFMKeyActionPressed) {
        ExampleApp *app = glfmGetUserData(display);
        switch (keyCode) {
            case GLFMKeyLeft:
                app->offsetX -= 0.1f;
                handled = true;
                break;
            case GLFMKeyRight:
                app->offsetX += 0.1f;
                handled = true;
                break;
            case GLFMKeyUp:
                app->offsetY += 0.1f;
                handled = true;
                break;
            case GLFMKeyDown:
                app->offsetY -= 0.1f;
                handled = true;
                break;
            default:
                break;
        }
    }
    return handled;
}

static void onSurfaceCreated(GLFMDisplay *display, int width, int height) {
    glViewport(0, 0, width, height);

    GLFMRenderingAPI api = glfmGetRenderingAPI(display);
    printf("Hello from GLFM! Using OpenGL %s\n",
           api == GLFMRenderingAPIOpenGLES32 ? "ES 3.2" :
           api == GLFMRenderingAPIOpenGLES31 ? "ES 3.1" :
           api == GLFMRenderingAPIOpenGLES3 ? "ES 3.0" : "ES 2.0");
}

static void onSurfaceDestroyed(GLFMDisplay *display) {
    // When the surface is destroyed, all existing GL resources are no longer valid.
```

```
    ExampleApp *app = glfmGetUserData(display);
    app->program = 0;
    app->vertexBuffer = 0;
}

static GLuint compileShader(GLenum type, const char *shaderName) {
    char fullPath[PATH_MAX];
    fc_resdir(fullPath, sizeof(fullPath));
    strncat(fullPath, shaderName, sizeof(fullPath) - strlen(fullPath) - 1);

    // Get shader string
    char *shaderString = NULL;
    FILE *shaderFile = fopen(fullPath, "rb");
    if (shaderFile) {
        fseek(shaderFile, 0, SEEK_END);
        long length = ftell(shaderFile);
        fseek(shaderFile, 0, SEEK_SET);

        shaderString = malloc(length + 1);
        if (shaderString) {
            fread(shaderString, length, 1, shaderFile);
            shaderString[length] = 0;
        }
        fclose(shaderFile);
    }
    if (!shaderString) {
        printf("Couldn't read file: %s\n", fullPath);
        return 0;
    }

    // Compile
    const char *constChaderString = shaderString;
    GLuint shader = glCreateShader(type);
    glShaderSource(shader, 1, &constChaderString, NULL);
    glCompileShader(shader);
    free(shaderString);

    // Check compile status
    GLint status;
    glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
    if (status == 0) {
        printf("Couldn't compile shader: %s\n", shaderName);
        GLint logLength;
        glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &logLength);
        if (logLength > 0) {
            GLchar *log = malloc(logLength);
            glGetShaderInfoLog(shader, logLength, &logLength, log);
            if (log[0] != 0) {
                printf("Shader log: %s\n", log);
            }
            free(log);
        }
        glDeleteShader(shader);
        shader = 0;
    }
    return shader;
}
```

```
static void onFrame(GLFMDisplay *display, double frameTime) {
    ExampleApp *app = glfmGetUserData(display);

    // Draw background
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    // Draw triangle
    if (app->program == 0) {
        GLuint vertShader = compileShader(GL_VERTEX_SHADER, "simple.vert");
        GLuint fragShader = compileShader(GL_FRAGMENT_SHADER, "simple.frag");
        if (vertShader == 0 || fragShader == 0) {
            glfmSetMainLoopFunc(display, NULL);
            return;
        }
        app->program = glCreateProgram();

        glAttachShader(app->program, vertShader);
        glAttachShader(app->program, fragShader);

        glBindAttribLocation(app->program, 0, "a_position");
        glBindAttribLocation(app->program, 1, "a_color");

        glLinkProgram(app->program);

        glDeleteShader(vertShader);
        glDeleteShader(fragShader);
    }
    glUseProgram(app->program);
    if (app->vertexBuffer == 0) {
        glGenBuffers(1, &app->vertexBuffer);
    }
    glBindBuffer(GL_ARRAY_BUFFER, app->vertexBuffer);
    const size_t stride = sizeof(GLfloat) * 6;
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, stride, (void *)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, stride, (void *)(sizeof(↙
        ↳ GLfloat) * 3));

    const GLfloat vertices[] = {
        // x,y,z, r,g,b
        app->offsetX + 0.0f, app->offsetY + 0.5f, 0.0,  1.0, 0.0, 0.0,
        app->offsetX - 0.5f, app->offsetY - 0.5f, 0.0,  0.0, 1.0, 0.0,
        app->offsetX + 0.5f, app->offsetY - 0.5f, 0.0,  0.0, 0.0, 1.0,
    };

    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
    glDrawArrays(GL_TRIANGLES, 0, 3);
}
```

37 example/src/test_pattern.c

```
// Draws a test pattern to check if framebuffer is scaled correctly.
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include "glfm.h"
#define FILE_COMPAT_ANDROID_ACTIVITY glfmAndroidGetActivity()
#include "file_compat.h"

typedef struct {
    GLuint textureId;
    GLuint textureProgram;
    GLuint textureVertexBuffer;
} TestPatternApp;

static GLuint createTestPatternTexture(uint32_t width, uint32_t height) {
    GLuint textureId = 0;
    uint32_t *data = malloc(width * height * sizeof(uint32_t));
    if (data) {
        uint32_t *out = data;
        for (uint32_t y = 0; y < height; y++) {
            *out++ = 0xff0000ff;
            if (y == 0 || y == height - 1) {
                for (uint32_t x = 1; x < width - 1; x++) {
                    *out++ = 0xff0000ff;
                }
            } else {
                for (uint32_t x = 1; x < width - 1; x++) {
                    *out++ = ((x & 1) == (y & 1)) ? 0xff000000 : 0xffffffff;
                }
            }
            *out++ = 0xff0000ff;
        }

        glGenTextures(1, &textureId);
        glBindTexture(GL_TEXTURE_2D, textureId);
        glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, ↙
            ↳ GL_UNSIGNED_BYTE, data);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

        free(data);
    }
    return textureId;
}

static void onSurfaceCreated(GLFMDisplay *display, int width, int height) {
    glViewport(0, 0, width, height);

    TestPatternApp *app = glfmGetUserData(display);
    if (app->textureId != 0) {
        glDeleteTextures(1, &app->textureId);
    }
    app->textureId = createTestPatternTexture(width, height);
    if (app->textureId != 0) {
        printf("Created test pattern %ix%i\n", width, height);
    }
}
```

```
60
static void onSurfaceDestroyed(GLFMDisplay *display) {
    // When the surface is destroyed, all existing GL resources are no longer ↙
        ↳ valid.
    TestPatternApp *app = glfmGetUserData(display);
    app->textureId = 0;
65  app->textureProgram = 0;
    app->textureVertexBuffer = 0;
}

static GLuint compileShader(GLenum type, const char *shaderName) {
70  char fullPath[PATH_MAX];
    fc_resdir(fullPath, sizeof(fullPath));
    strncat(fullPath, shaderName, sizeof(fullPath) - strlen(fullPath) - 1);

    // Get shader string
75  char *shaderString = NULL;
    FILE *shaderFile = fopen(fullPath, "rb");
    if (shaderFile) {
        fseek(shaderFile, 0, SEEK_END);
        long length = ftell(shaderFile);
80      fseek(shaderFile, 0, SEEK_SET);

        shaderString = malloc(length + 1);
        if (shaderString) {
            fread(shaderString, length, 1, shaderFile);
85          shaderString[length] = 0;
        }
        fclose(shaderFile);
    }
    if (!shaderString) {
90      printf("Couldn't read file: %s\n", fullPath);
        return 0;
    }

    // Compile
95  const char *constChaderString = shaderString;
    GLuint shader = glCreateShader(type);
    glShaderSource(shader, 1, &constChaderString, NULL);
    glCompileShader(shader);
    free(shaderString);
100
    // Check compile status
    GLint status;
    glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
    if (status == 0) {
105     printf("Couldn't compile shader: %s\n", shaderName);
        GLint logLength;
        glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &logLength);
        if (logLength > 0) {
            GLchar *log = malloc(logLength);
110         glGetShaderInfoLog(shader, logLength, &logLength, log);
            if (log[0] != 0) {
                printf("Shader log: %s\n", log);
            }
            free(log);
115     }
```

```
        glDeleteShader(shader);
        shader = 0;
    }
    return shader;
}

static void onFrame(GLFMDisplay *display, double frameTime) {
    TestPatternApp *app = glfmGetUserData(display);

    if (app->textureId == 0) {
        glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);
        return;
    }

    if (app->textureProgram == 0) {
        GLuint vertShader = compileShader(GL_VERTEX_SHADER, "texture.vert");
        GLuint fragShader = compileShader(GL_FRAGMENT_SHADER, "texture.frag");
        if (vertShader == 0 || fragShader == 0) {
            glfmSetMainLoopFunc(display, NULL);
            return;
        }

        app->textureProgram = glCreateProgram();

        glAttachShader(app->textureProgram, vertShader);
        glAttachShader(app->textureProgram, fragShader);

        glBindAttribLocation(app->textureProgram, 0, "position");
        glBindAttribLocation(app->textureProgram, 1, "texCoord");

        glLinkProgram(app->textureProgram);

        glDeleteShader(vertShader);
        glDeleteShader(fragShader);
    }
    glUseProgram(app->textureProgram);
    if (app->textureVertexBuffer == 0) {
        glGenBuffers(1, &app->textureVertexBuffer);
    }
    glBindBuffer(GL_ARRAY_BUFFER, app->textureVertexBuffer);
    const size_t stride = sizeof(GLfloat) * 4;
    const size_t textureCoordsOffset = sizeof(GLfloat) * 2;
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, stride, (void *)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, stride, (void *)↙
        ↳ textureCoordsOffset);

    const GLfloat vertices[] = {
        // viewX, viewY, textureX, textureY
        -1, -1, 0, 0,
         1, -1, 1, 0,
        -1,  1, 0, 1,
         1,  1, 1, 1,
    };
```

```
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_DYNAMIC_DRAW);
    glBindTexture(GL_TEXTURE_2D, app->textureId);
    glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
}

void glfmMain(GLFMDisplay *display) {
    TestPatternApp *app = calloc(1, sizeof(TestPatternApp));

    glfmSetDisplayConfig(display,
                         GLFMRenderingAPIOpenGLES2,
                         GLFMColorFormatRGBA8888,
                         GLFMDepthFormatNone,
                         GLFMStencilFormatNone,
                         GLFMMultisampleNone);
    glfmSetUserData(display, app);
    glfmSetSurfaceCreatedFunc(display, onSurfaceCreated);
    glfmSetSurfaceResizedFunc(display, onSurfaceCreated);
    glfmSetSurfaceDestroyedFunc(display, onSurfaceDestroyed);
    glfmSetMainLoopFunc(display, onFrame);
}
```

38 .gitignore

```
build
```

39 include/glfm.h

```
/*
 GLFM
 https://github.com/brackeen/glfm
 Copyright (c) 2014-2017 David Brackeen

 This software is provided 'as-is', without any express or implied warranty.
 In no event will the authors be held liable for any damages arising from the
 use of this software. Permission is granted to anyone to use this software
 for any purpose, including commercial applications, and to alter it and
 redistribute it freely, subject to the following restrictions:

 1. The origin of this software must not be misrepresented; you must not
    claim that you wrote the original software. If you use this software in a
    product, an acknowledgment in the product documentation would be appreciated
    but is not required.
 2. Altered source versions must be plainly marked as such, and must not be
    misrepresented as being the original software.
 3. This notice may not be removed or altered from any source distribution.
 */

#ifndef GLFM_H
#define GLFM_H

#define GLFM_VERSION_MAJOR 0
#define GLFM_VERSION_MINOR 9
#define GLFM_VERSION_REVISION 0

// One of these will be defined:
// GLFM_PLATFORM_IOS
```

```
30  // GLFM_PLATFORM_TVOS
    // GLFM_PLATFORM_ANDROID
    // GLFM_PLATFORM_EMSCRIPTEN

    #if defined(__ANDROID__)
35    #define GLFM_PLATFORM_ANDROID
    #elif defined(__EMSCRIPTEN__)
      #define GLFM_PLATFORM_EMSCRIPTEN
    #elif defined(__APPLE__)
      #include <TargetConditionals.h>
40    #if TARGET_OS_IOS
        #define GLFM_PLATFORM_IOS
      #elif TARGET_OS_TV
        #define GLFM_PLATFORM_TVOS
      #else
45      #error Unknown Apple platform
      #endif
    #else
      #error Unknown platform
    #endif
50
    // OpenGL ES includes

    #if defined(GLFM_INCLUDE_ES32)
      #if defined(GLFM_PLATFORM_IOS) || defined(GLFM_PLATFORM_TVOS)
55      #error No OpenGL ES 3.2 support in iOS
      #elif defined(GLFM_PLATFORM_EMSCRIPTEN)
        #error No OpenGL ES 3.2 support in WebGL
      #else
        #include <GLES3/gl32.h>
60      #include <GLES3/gl3ext.h>
      #endif
    #elif defined(GLFM_INCLUDE_ES31)
      #if defined(GLFM_PLATFORM_IOS) || defined(GLFM_PLATFORM_TVOS)
        #error No OpenGL ES 3.1 support in iOS
65    #elif defined(GLFM_PLATFORM_EMSCRIPTEN)
        #error No OpenGL ES 3.1 support in WebGL
      #else
        #include <GLES3/gl31.h>
        #include <GLES3/gl3ext.h>
70    #endif
    #elif defined(GLFM_INCLUDE_ES3)
      #if defined(GLFM_PLATFORM_IOS) || defined(GLFM_PLATFORM_TVOS)
        #include <OpenGLES/ES3/gl.h>
        #include <OpenGLES/ES3/glext.h>
75    #elif defined(GLFM_PLATFORM_EMSCRIPTEN)
        #include <GLES3/gl3.h>
        #include <GLES3/gl2ext.h>
      #else
        #include <GLES3/gl3.h>
80      #include <GLES3/gl3ext.h>
      #endif
    #elif !defined(GLFM_INCLUDE_NONE)
      #if defined(GLFM_PLATFORM_IOS) || defined(GLFM_PLATFORM_TVOS)
        #include <OpenGLES/ES2/gl.h>
85      #include <OpenGLES/ES2/glext.h>
      #else
```

```
    #include <GLES2/gl2.h>
    #include <GLES2/gl2ext.h>
  #endif
#endif

#include <stdbool.h>

#ifdef __cplusplus
extern "C" {
#endif

// MARK: Enums

typedef enum {
    GLFMRenderingAPIOpenGLES2,
    GLFMRenderingAPIOpenGLES3,
    GLFMRenderingAPIOpenGLES31,
    GLFMRenderingAPIOpenGLES32,
} GLFMRenderingAPI;

typedef enum {
    GLFMColorFormatRGBA8888,
    GLFMColorFormatRGB565,
} GLFMColorFormat;

typedef enum {
    GLFMDepthFormatNone,
    GLFMDepthFormat16,
    GLFMDepthFormat24,
} GLFMDepthFormat;

typedef enum {
    GLFMStencilFormatNone,
    GLFMStencilFormat8,
} GLFMStencilFormat;

typedef enum {
    GLFMMultisampleNone,
    GLFMMultisample4X,
} GLFMMultisample;

/// GLFMUserInterfaceChrome defines whether system UI chrome (status bar, navigation bar) is shown.
/// This value is ignored on Emscripten.
/// GLFMUserInterfaceChromeNavigation (default)
///  - Android: Show the navigation bar
///  - iOS: Show the home indicator on iPhone X
/// GLFMUserInterfaceChromeNavigationAndStatusBar:
///  - Android: Show the navigation bar and status bar
///  - iOS: Show status bar, and show the home indicator on iPhone X
/// GLFMUserInterfaceChromeFullscreen
///  - Android 2.3: Fullscreen
///  - Android 4.0 - 4.3: Navigation bar dimmed
///  - Android 4.4: Fullscreen immersive mode
///  - iOS: Fullscreen
typedef enum {
    GLFMUserInterfaceChromeNavigation,
```

```
    GLFMUserInterfaceChromeNavigationAndStatusBar,
    GLFMUserInterfaceChromeFullscreen,
} GLFMUserInterfaceChrome;

typedef enum {
    GLFMUserInterfaceOrientationAny,
    GLFMUserInterfaceOrientationPortrait,
    GLFMUserInterfaceOrientationLandscape,
} GLFMUserInterfaceOrientation;

typedef enum {
    GLFMTouchPhaseHover,
    GLFMTouchPhaseBegan,
    GLFMTouchPhaseMoved,
    GLFMTouchPhaseEnded,
    GLFMTouchPhaseCancelled,
} GLFMTouchPhase;

typedef enum {
    GLFMMouseCursorAuto,
    GLFMMouseCursorNone,
    GLFMMouseCursorDefault,
    GLFMMouseCursorPointer,
    GLFMMouseCursorCrosshair,
    GLFMMouseCursorText
} GLFMMouseCursor;

typedef enum {
    GLFMKeyBackspace = 0x08,
    GLFMKeyTab = 0x09,
    GLFMKeyEnter = 0x0d,
    GLFMKeyEscape = 0x1b,
    GLFMKeySpace = 0x20,
    GLFMKeyLeft = 0x25,
    GLFMKeyUp = 0x26,
    GLFMKeyRight = 0x27,
    GLFMKeyDown = 0x28,
    GLFMKeyNavBack = 0x1000,
    GLFMKeyNavMenu = 0x1001,
    GLFMKeyNavSelect = 0x1002,
    GLFMKeyPlayPause = 0x2000,
} GLFMKey;

typedef enum {
    GLFMKeyModifierShift = (1 << 0),
    GLFMKeyModifierCtrl = (1 << 1),
    GLFMKeyModifierAlt = (1 << 2),
    GLFMKeyModifierMeta = (1 << 3),
} GLFMKeyModifier;

typedef enum {
    GLFMKeyActionPressed,
    GLFMKeyActionRepeated,
    GLFMKeyActionReleased,
} GLFMKeyAction;

// MARK: Structs and function pointers
```

```
200
    typedef struct GLFMDisplay GLFMDisplay;

    /// Function pointer returned from glfmGetProcAddress
    typedef void (*GLFMProc)(void);
205
    /// Main loop callback function. The frame time is in seconds, and is not ↙
        ↳ related to wall time.
    typedef void (*GLFMMainLoopFunc)(GLFMDisplay *display, double frameTime);

    /// Callback function for mouse or touch events. The (x,y) values are in pixels.
210 /// The function should return true if the event was handled, and false ↙
        ↳ otherwise.
    typedef bool (*GLFMTouchFunc)(GLFMDisplay *display, int touch, GLFMTouchPhase ↙
        ↳ phase,
                                  double x, double y);

    /// Callback function for key events.
215 /// The function should return true if the event was handled, and false ↙
        ↳ otherwise.
    typedef bool (*GLFMKeyFunc)(GLFMDisplay *display, GLFMKey keyCode, GLFMKeyAction↙
        ↳  action,
                                int modifiers);

    /// Callback function for character input events.
220 typedef void (*GLFMCharFunc)(GLFMDisplay *display, const char *utf8, int ↙
        ↳ modifiers);

    /// Callback function for keyboard visibility, in pixels.
    typedef void (*GLFMKeyboardVisibilityChangedFunc)(GLFMDisplay *display, bool ↙
        ↳ visible,
                                                      double x, double y, double ↙
                                                          ↳ width, double height);
225
    /// Callback when the surface could not be created.
    typedef void (*GLFMSurfaceErrorFunc)(GLFMDisplay *display, const char *message);

    /// Callback function when the OpenGL surface is created
230 typedef void (*GLFMSurfaceCreatedFunc)(GLFMDisplay *display, int width, int ↙
        ↳ height);

    /// Callback function when the OpenGL surface is resized (or rotated).
    typedef void (*GLFMSurfaceResizedFunc)(GLFMDisplay *display, int width, int ↙
        ↳ height);

235 /// Callback function when the OpenGL surface is destroyed.
    typedef void (*GLFMSurfaceDestroyedFunc)(GLFMDisplay *display);

    /// Callback function when the system recieves a low memory warning.
    typedef void (*GLFMMemoryWarningFunc)(GLFMDisplay *display);
240
    typedef void (*GLFMAppFocusFunc)(GLFMDisplay *display, bool focused);

    // MARK: Functions

245 /// Main entry point for the app, where the display can be initialized and the ↙
        ↳ GLFMMainLoopFunc
```

```
/// can be set.
extern void glfmMain(GLFMDisplay *display);

/// Init the display condifuration. Should only be called in glfmMain.
/// If the device does not support the preferred rendering API, the next available rendering API is
/// chosen (OpenGL ES 3.0 if OpenGL ES 3.1 is not available, and OpenGL ES 2.0 if OpenGL ES 3.0 is
/// not available). Call glfmGetRenderingAPI in the GLFMSurfaceCreatedFunc to see which rendering
/// API was chosen.
void glfmSetDisplayConfig(GLFMDisplay *display,
                          GLFMRenderingAPI preferredAPI,
                          GLFMColorFormat colorFormat,
                          GLFMDepthFormat depthFormat,
                          GLFMStencilFormat stencilFormat,
                          GLFMMultisample multisample);

void glfmSetUserData(GLFMDisplay *display, void *userData);

void *glfmGetUserData(GLFMDisplay *display);

/// Sets the allowed user interface orientations
void glfmSetUserInterfaceOrientation(GLFMDisplay *display,
                                     GLFMUserInterfaceOrientation allowedOrientations);

/// Returns the allowed user interface orientations
GLFMUserInterfaceOrientation glfmGetUserInterfaceOrientation(GLFMDisplay *display);

/// Sets whether multitouch input is enabled. By default, multitouch is disabled.
void glfmSetMultitouchEnabled(GLFMDisplay *display, bool multitouchEnabled);

/// Gets whether multitouch input is enabled. By default, multitouch is disabled.
bool glfmGetMultitouchEnabled(GLFMDisplay *display);

/// Gets the display size, in pixels.
void glfmGetDisplaySize(GLFMDisplay *display, int *width, int *height);

/// Gets the display scale. On Apple devices, the value will be 1.0 for non-retina displays and 2.0
/// for retina.
double glfmGetDisplayScale(GLFMDisplay *display);

/// Gets the chrome insets, in pixels (AKA "safe area insets" in iOS). This is the space taken
/// on the outer edges of the display by status bars, navigation bars, and other UI elements.
void glfmGetDisplayChromeInsets(GLFMDisplay *display, double *top, double *right, double *bottom,
                                double *left);

/// Gets the user interface chrome.
GLFMUserInterfaceChrome glfmGetDisplayChrome(GLFMDisplay *display);
```

```
    /// Sets the user interface chrome.
    /// On Emscripten, to switch to fullscreen, this function must be called from an↙
        ↪ user-generated event handler.
295 void glfmSetDisplayChrome(GLFMDisplay *display, GLFMUserInterfaceChrome uiChrome↙
        ↪ );

    /// Gets the rendering API of the display. The return value is not valid until ↙
        ↪ the surface is
    /// created. Defaults to GLFMRenderingAPIOpenGLES2.
    GLFMRenderingAPI glfmGetRenderingAPI(GLFMDisplay *display);
300
    /// Gets whether the display has touch capabilities.
    bool glfmHasTouch(GLFMDisplay *display);

    /// Sets the mouse cursor (only on platforms with a mouse)
305 void glfmSetMouseCursor(GLFMDisplay *display, GLFMMouseCursor mouseCursor);

    /// Checks if a named OpenGL extension is supported
    bool glfmExtensionSupported(const char *extension);

310 /// Gets the address of the specified function.
    GLFMProc glfmGetProcAddress(const char *functionName);

    /// Sets the function to call before each frame is displayed.
    void glfmSetMainLoopFunc(GLFMDisplay *display, GLFMMainLoopFunc mainLoopFunc);
315
    /// Sets the function to call when a mouse or touch event occurs.
    void glfmSetTouchFunc(GLFMDisplay *display, GLFMTouchFunc touchFunc);

    /// Sets the function to call when a key event occurs.
320 /// Note, on iOS, only pressed events are sent (no repeated or released events) ↙
        ↪ and with no
    /// modifiers.
    void glfmSetKeyFunc(GLFMDisplay *display, GLFMKeyFunc keyFunc);

    /// Sets the function to call when character input events occur.
325 void glfmSetCharFunc(GLFMDisplay *display, GLFMCharFunc charFunc);

    /// Sets the function to call when the surface could not be created.
    /// For example, the browser does not support WebGL.
    void glfmSetSurfaceErrorFunc(GLFMDisplay *display, GLFMSurfaceErrorFunc ↙
        ↪ surfaceErrorFunc);
330
    /// Sets the function to call when the surface was created.
    void glfmSetSurfaceCreatedFunc(GLFMDisplay *display, GLFMSurfaceCreatedFunc ↙
        ↪ surfaceCreatedFunc);

    /// Sets the function to call when the surface was resized (or rotated).
335 void glfmSetSurfaceResizedFunc(GLFMDisplay *display, GLFMSurfaceResizedFunc ↙
        ↪ surfaceResizedFunc);

    /// Sets the function to call when the surface was destroyed. For example, ↙
        ↪ OpenGL context loss.
    /// All OpenGL resources should be deleted in this call.
    void glfmSetSurfaceDestroyedFunc(GLFMDisplay *display,
340                                  GLFMSurfaceDestroyedFunc surfaceDestroyedFunc);
```

```
void glfmSetMemoryWarningFunc(GLFMDisplay *display, GLFMMemoryWarningFunc
    ↪ lowMemoryFunc);

void glfmSetAppFocusFunc(GLFMDisplay *display, GLFMAppFocusFunc focusFunc);

/// Requests to show or hide the onscreen virtual keyboard. On Emscripten, this
    ↪ function does
/// nothing.
void glfmSetKeyboardVisible(GLFMDisplay *display, bool visible);

/// Returns true if the virtual keyboard is currently visible.
bool glfmIsKeyboardVisible(GLFMDisplay *display);

/// Sets the function to call when the virtual keyboard changes visibility or
    ↪ changes bounds.
void glfmSetKeyboardVisibilityChangedFunc(GLFMDisplay *display,
                                          GLFMKeyboardVisibilityChangedFunc
                                              ↪ visibilityChangedFunc);

#if defined(GLFM_PLATFORM_ANDROID)

#include <android/native_activity.h>

ANativeActivity *glfmAndroidGetActivity(void);

#endif // GLFM_PLATFORM_ANDROID

#ifdef __cplusplus
}
#endif

#endif
```

40 LICENSE

```
Copyright (c) 2014-2017 David Brackeen

This software is provided 'as-is', without any express or implied
warranty. In no event will the authors be held liable for any damages
arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not
   claim that you wrote the original software. If you use this software
   in a product, an acknowledgment in the product documentation would
   be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not
   be misrepresented as being the original software.

3. This notice may not be removed or altered from any source
   distribution.
```

41 README.graphgrow.md

```
This is:

- the GLFM git repository, with an Android build system

- liblo-0.29, with a simplified CMake build system and minor patches

- graphgrow-iface.cpp from the graphgrow git repository, modified to
  work with GLFM instead of GLFW, and to send OSC to network in a non-UI
  thread
```

42 README.md

```
# GLFM
Write OpenGL ES code in C/C++ without writing platform-specific code.

GLFM is an OpenGL ES layer for mobile devices and the web. GLFM supplies an ↙
    ↳ OpenGL ES context and input events. It is largely inspired by [GLFW](https↙
    ↳ ://github.com/glfw/glfw).

GLFM is written in C and runs on iOS 8, tvOS 9, Android 2.3.3 (API 10), and ↙
    ↳ WebGL 1.0 (via [Emscripten](https://github.com/kripken/emscripten)).

## Features
* OpenGL ES 2.0, 3.0, 3.1, and 3.2 display setup.
* Retina / high-DPI support.
* Touch and keyboard events.
* Events for application state and context loss.

## Non-goals
GLFM is limited in scope, and isn't designed to provide everything needed for an↙
    ↳  app. For example, GLFM doesn't provide (and will never provide) the ↙
    ↳ following:

* No image loading.
* No text rendering.
* No audio.
* No menus, UI toolkit, or scene graph.
* No integration with other mobile features like web views, maps, or game scores↙
    ↳ .

Instead, GLFM can be used with other cross-platform libraries that provide what ↙
    ↳ an app needs.

## Example
This example initializes the display in `glfmMain()` and draws a triangle in `↙
    ↳ onFrame()`. A more detailed example is available [here](example/src/main.c↙
    ↳ ).

```C
#include "glfm.h"
#include <string.h>

static GLint program = 0;
static GLuint vertexBuffer = 0;
```
```

```
35  static void onFrame(GLFMDisplay *display, const double frameTime);
    static void onSurfaceCreated(GLFMDisplay *display, const int width, const int ↙
        ↳ height);
    static void onSurfaceDestroyed(GLFMDisplay *display);

    void glfmMain(GLFMDisplay *display) {
40      glfmSetDisplayConfig(display,
                             GLFMRenderingAPIOpenGLES2,
                             GLFMColorFormatRGBA8888,
                             GLFMDepthFormatNone,
                             GLFMStencilFormatNone,
45                           GLFMMultisampleNone);
        glfmSetSurfaceCreatedFunc(display, onSurfaceCreated);
        glfmSetSurfaceResizedFunc(display, onSurfaceCreated);
        glfmSetSurfaceDestroyedFunc(display, onSurfaceDestroyed);
        glfmSetMainLoopFunc(display, onFrame);
50  }

    static void onSurfaceCreated(GLFMDisplay *display, const int width, const int ↙
        ↳ height) {
        glViewport(0, 0, width, height);
    }
55
    static void onSurfaceDestroyed(GLFMDisplay *display) {
        // When the surface is destroyed, all existing GL resources are no longer ↙
            ↳ valid.
        program = 0;
        vertexBuffer = 0;
60  }

    static GLuint compileShader(const GLenum type, const GLchar *shaderString) {
        const GLint shaderLength = (GLint)strlen(shaderString);
        GLuint shader = glCreateShader(type);
65      glShaderSource(shader, 1, &shaderString, &shaderLength);
        glCompileShader(shader);
        return shader;
    }

70  static void onFrame(GLFMDisplay *display, const double frameTime) {
        if (program == 0) {
            const GLchar *vertexShader =
                "attribute highp vec4 position;\n"
                "void main() {\n"
75              "    gl_Position = position;\n"
                "}";

            const GLchar *fragmentShader =
                "void main() {\n"
80              "  gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);\n"
                "}";

            program = glCreateProgram();
            GLuint vertShader = compileShader(GL_VERTEX_SHADER, vertexShader);
85          GLuint fragShader = compileShader(GL_FRAGMENT_SHADER, fragmentShader);

            glAttachShader(program, vertShader);
            glAttachShader(program, fragShader);
```

```
90          glLinkProgram(program);

            glDeleteShader(vertShader);
            glDeleteShader(fragShader);
        }
95      if (vertexBuffer == 0) {
            const GLfloat vertices[] = {
                 0.0,  0.5, 0.0,
                -0.5, -0.5, 0.0,
                 0.5, -0.5, 0.0,
100         };
            glGenBuffers(1, &vertexBuffer);
            glBindBuffer(GL_ARRAY_BUFFER, vertexBuffer);
            glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW↙
                ↳ );
        }
105
        glClearColor(0.4f, 0.0f, 0.6f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        glUseProgram(program);
110     glBindBuffer(GL_ARRAY_BUFFER, vertexBuffer);

        glEnableVertexAttribArray(0);
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
        glDrawArrays(GL_TRIANGLES, 0, 3);
115 }
    ```
    ## API
    See [glfm.h](include/glfm.h)

120 ## Build requirements
    * iOS: Xcode 9.0
    * Android: Android Studio 3.0, SDK 26, NDK Bundle 16.
    * WebGL: Emscripten 1.35.0

125 ## Use GLFM in an existing project

    1. Remove the project's existing `void main()` function, if any.
    2. Add the GLFM source files (in `include` and `src`).
    3. Include a `void glfmMain(GLFMDisplay *display)` function in a C/C++ file.
130
    ## Build the example GLFM projects
    Use the `CMakeLists.txt` file with the `-DGLFM_BUILD_EXAMPLE=ON` option to build↙
        ↳ the example projects.

    ### Xcode 9.0
135 ```Shell
    mkdir -p build/ios
    cd build/ios
    cmake -DGLFM_BUILD_EXAMPLE=ON -G Xcode ../..
    open GLFM.xcodeproj
140 ```
    Switch to the `glfm_example` target and run on the simulator or a device.

    ### Emscripten
```
```

```
Assuming `EMSCRIPTEN_ROOT_PATH` points to active installed version of Emscripten
    ↳ .
```shell
mkdir -p build/emscripten
cd build/emscripten
cmake -DGLFM_BUILD_EXAMPLE=ON -DCMAKE_TOOLCHAIN_FILE=$EMSCRIPTEN_ROOT_PATH/cmake
    ↳ /Modules/Platform/Emscripten.cmake -DCMAKE_BUILD_TYPE=MinSizeRel ../..
cmake --build .
```
If you're opening files locally in Chrome, you may need to [enable local file
    ↳ access](http://stackoverflow.com/a/18587027). Instead, you could use
    ↳ Firefox, which doesn't have this restriction.

Android Studio 3.0
There is no CMake generator for Android Studio projects, but you can include `
    ↳ CMakeLists.txt` in a new or existing project.

The `AndroidManifest.xml`:
```XML
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.brackeen.glfmexample">

    <uses-feature android:glEsVersion="0x00020000" android:required="true" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true">
        <activity android:name="android.app.NativeActivity"
                  android:configChanges="orientation|screenLayout|screenSize|
                      ↳ keyboardHidden|keyboard">
            <meta-data
                android:name="android.app.lib_name"
                android:value="glfm_example" />
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```
And the `app/build.gradle`:
```Gradle
apply plugin: 'com.android.application'

android {
    compileSdkVersion 26
    defaultConfig {
        applicationId "com.brackeen.glfmexample"
        minSdkVersion 10
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
```
```

```
        externalNativeBuild {
            cmake {
                arguments "-DGLFM_BUILD_EXAMPLE=ON"
            }
        }
    }
    sourceSets.main {
        assets.srcDirs = ["../../../../example/assets"]
    }
    externalNativeBuild {
        cmake {
            path "../../../../CMakeLists.txt"
        }
    }
}
```

Future ideas
* Accelerometer and gyroscope input.
* Gamepad / MFi controller input.

Caveats
* OpenGL ES 3.1 and 3.2 support is only available in Android, and the GLFM implementation is currently untested.
* GLFM is not thread-safe. All GLFM functions must be called on the main thread (that is, from `glfmMain` or from the callback functions).
* On iOS, character input works great, but keyboard events are not ideal. Using the keyboard (on an iOS device via Bluetooth keyboard or on the simulator via a Mac's keyboard), only a few keys are detected (arrows keys and the escape key), and key release events are not reported.
* On Android, keyboard events work great, but character events are not ideal. Some special characters, like emoji characters, will not work. This is due to an issue in the NDK.
* Orientation lock probably doesn't work on HTML5.

Questions
What IDE should I use? Why is there no desktop implementation?
Use Xcode or Android Studio. For desktop, use [GLFW](https://github.com/glfw/glfw) with the IDE of your choice.

If you prefer not using the mobile simulators for everyday development, a good solution is to use GLFW instead, and then later port your app to GLFM. Not all OpenGL calls will port to OpenGL ES perfectly, but for maximum OpenGL portability, use OpenGL 3.2 Core Profile on desktop and OpenGL ES 2.0 on mobile.

Moving forward, GLFM APIs will look more like GLFW's, so porting will get easier as GLFM development continues.

Why is the entry point `glfmMain()` and not `main()`?

Otherwise, it wouldn't work on iOS. To initialize the Objective-C environment, the `main()` function must create an autorelease pool and call the `UIApplicationMain()` function, which *never returns*. On iOS, GLFM doesn't call `glfmMain()` until after the `UIApplicationDelegate` and `UIViewController` are initialized.

```
235 **Why is GLFM event-driven? Why does GLFM take over the main loop?**

Otherwise, it wouldn't work on iOS (see above) or on HTML5, which is event-↙
    ↪ driven.
```

43 src/glfm_platform_android.c

```
/*
 GLFM
 https://github.com/brackeen/glfm
 Copyright (c) 2014-2017 David Brackeen
5
 This software is provided 'as-is', without any express or implied warranty.
 In no event will the authors be held liable for any damages arising from the
 use of this software. Permission is granted to anyone to use this software
 for any purpose, including commercial applications, and to alter it and
10 redistribute it freely, subject to the following restrictions:

 1. The origin of this software must not be misrepresented; you must not
    claim that you wrote the original software. If you use this software in a
    product, an acknowledgment in the product documentation would be appreciated
15  but is not required.
 2. Altered source versions must be plainly marked as such, and must not be
    misrepresented as being the original software.
 3. This notice may not be removed or altered from any source distribution.
 */
20
#include "glfm.h"

#ifdef GLFM_PLATFORM_ANDROID

25 #include "android_native_app_glue.h"
#include "glfm_platform.h"
#include <EGL/egl.h>
#include <android/log.h>
#include <android/window.h>
30 #include <dlfcn.h>
#include <unistd.h>

#ifdef NDEBUG
#define LOG_DEBUG(...) do { } while (0)
35 #else
#define LOG_DEBUG(...) __android_log_print(ANDROID_LOG_INFO, "GLFM", __VA_ARGS__↙
    ↪ )
#endif

//#define LOG_LIFECYCLE(...) __android_log_print(ANDROID_LOG_INFO, "GLFM", ↙
    ↪ __VA_ARGS__)
40 #define LOG_LIFECYCLE(...) do { } while (0)

// MARK: Time utils

static struct timespec _glfmTimeNow() {
45     struct timespec t;
    clock_gettime(CLOCK_MONOTONIC_RAW, &t);
    return t;
}
```

```
50  static double _glfmTimeSeconds(struct timespec t) {
        return t.tv_sec + (double)t.tv_nsec / 1e9;
    }

    static struct timespec _glfmTimeSubstract(struct timespec a, struct timespec b) ↙
        ↳ {
55      struct timespec result;
        if (b.tv_nsec > a.tv_nsec) {
            result.tv_sec = a.tv_sec - b.tv_sec - 1;
            result.tv_nsec = 1000000000 - b.tv_nsec + a.tv_nsec;
        } else {
60          result.tv_sec = a.tv_sec - b.tv_sec;
            result.tv_nsec = a.tv_nsec - b.tv_nsec;
        }
        return result;
    }
65
    // MARK: Platform data (global singleton)

    #define MAX_SIMULTANEOUS_TOUCHES 5

70  typedef struct {
        struct android_app *app;

        bool multitouchEnabled;

75      ARect keyboardFrame;
        bool keyboardVisible;

        struct timespec initTime;
        bool animating;
80      bool hasInited;

        EGLDisplay eglDisplay;
        EGLSurface eglSurface;
        EGLConfig eglConfig;
85      EGLContext eglContext;
        bool eglContextCurrent;

        int32_t width;
        int32_t height;
90      double scale;

        GLFMDisplay *display;
        GLFMRenderingAPI renderingAPI;

95      JNIEnv *jniEnv;
    } GLFMPlatformData;

    static GLFMPlatformData *platformDataGlobal = NULL;

100 // MARK: JNI code

    #define _glfmWasJavaExceptionThrown() \
        ((*jni)->ExceptionCheck(jni) ? ((*jni)->ExceptionClear(jni), true) : false)
```

```
105 #define _glfmClearJavaException() \
        if ((*jni)->ExceptionCheck(jni)) { \
            (*jni)->ExceptionClear(jni); \
        }

110 static jmethodID _glfmGetJavaMethodID(JNIEnv *jni, jobject object, const char *↙
        ↳ name,
                                      const char *sig) {
        if (object) {
            jclass class = (*jni)->GetObjectClass(jni, object);
            jmethodID methodID = (*jni)->GetMethodID(jni, class, name, sig);
115         (*jni)->DeleteLocalRef(jni, class);
            return _glfmWasJavaExceptionThrown() ? NULL : methodID;
        } else {
            return NULL;
        }
120 }

    static jfieldID _glfmGetJavaFieldID(JNIEnv *jni, jobject object, const char *↙
        ↳ name, const char *sig) {
        if (object) {
            jclass class = (*jni)->GetObjectClass(jni, object);
125         jfieldID fieldID = (*jni)->GetFieldID(jni, class, name, sig);
            (*jni)->DeleteLocalRef(jni, class);
            return _glfmWasJavaExceptionThrown() ? NULL : fieldID;
        } else {
            return NULL;
130     }
    }

    static jfieldID _glfmGetJavaStaticFieldID(JNIEnv *jni, jclass class, const char ↙
        ↳ *name,
                                          const char *sig) {
135     if (class) {
            jfieldID fieldID = (*jni)->GetStaticFieldID(jni, class, name, sig);
            return _glfmWasJavaExceptionThrown() ? NULL : fieldID;
        } else {
            return NULL;
140     }
    }

    #define _glfmCallJavaMethod(jni, object, methodName, methodSig, returnType) \
        (*jni)->Call##returnType##Method(jni, object, \
145         _glfmGetJavaMethodID(jni, object, methodName, methodSig))

    #define _glfmCallJavaMethodWithArgs(jni, object, methodName, methodSig, ↙
        ↳ returnType, ...) \
        (*jni)->Call##returnType##Method(jni, object, \
            _glfmGetJavaMethodID(jni, object, methodName, methodSig), __VA_ARGS__)
150
    #define _glfmGetJavaField(jni, object, fieldName, fieldSig, fieldType) \
        (*jni)->Get##fieldType##Field(jni, object, \
            _glfmGetJavaFieldID(jni, object, fieldName, fieldSig))

155 #define _glfmGetJavaStaticField(jni, class, fieldName, fieldSig, fieldType) \
        (*jni)->GetStatic##fieldType##Field(jni, class, \
            _glfmGetJavaStaticFieldID(jni, class, fieldName, fieldSig))
```

```
static void _glfmSetOrientation(struct android_app *app) {
    static const int ActivityInfo_SCREEN_ORIENTATION_SENSOR = 0x00000004;
    static const int ActivityInfo_SCREEN_ORIENTATION_SENSOR_LANDSCAPE = 0x00000006;
    static const int ActivityInfo_SCREEN_ORIENTATION_SENSOR_PORTRAIT = 0x00000007;

    GLFMPlatformData *platformData = (GLFMPlatformData *)app->userData;
    int orientation;
    switch (platformData->display->allowedOrientations) {
        case GLFMUserInterfaceOrientationPortrait:
            orientation = ActivityInfo_SCREEN_ORIENTATION_SENSOR_PORTRAIT;
            break;
        case GLFMUserInterfaceOrientationLandscape:
            orientation = ActivityInfo_SCREEN_ORIENTATION_SENSOR_LANDSCAPE;
            break;
        case GLFMUserInterfaceOrientationAny:
        default:
            orientation = ActivityInfo_SCREEN_ORIENTATION_SENSOR;
            break;
    }

    JNIEnv *jni = platformData->jniEnv;
    if ((*jni)->ExceptionCheck(jni)) {
        return;
    }

    _glfmCallJavaMethodWithArgs(jni, app->activity->clazz, "setRequestedOrientation", "(I)V", Void,
                                orientation);
    _glfmClearJavaException()
}

static jobject _glfmGetDecorView(struct android_app *app) {
    GLFMPlatformData *platformData = (GLFMPlatformData *)app->userData;
    JNIEnv *jni = platformData->jniEnv;
    if ((*jni)->ExceptionCheck(jni)) {
        return NULL;
    }
    jobject window = _glfmCallJavaMethod(jni, app->activity->clazz, "getWindow",
                                         "()Landroid/view/Window;", Object);
    if (!window || _glfmWasJavaExceptionThrown()) {
        return NULL;
    }
    jobject decorView = _glfmCallJavaMethod(jni, window, "getDecorView", "()Landroid/view/View;",
                                            Object);
    (*jni)->DeleteLocalRef(jni, window);
    return _glfmWasJavaExceptionThrown() ? NULL : decorView;
}

static void _glfmSetFullScreen(struct android_app *app, GLFMUserInterfaceChrome uiChrome) {
    static const int View_STATUS_BAR_HIDDEN = 0x00000001;
    static const int View_SYSTEM_UI_FLAG_LOW_PROFILE = 0x00000001;
    static const int View_SYSTEM_UI_FLAG_HIDE_NAVIGATION = 0x00000002;
```

```
210     static const int View_SYSTEM_UI_FLAG_FULLSCREEN = 0x00000004;
        static const int View_SYSTEM_UI_FLAG_LAYOUT_STABLE = 0x00000100;
        static const int View_SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION = 0x00000200;
        static const int View_SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN = 0x00000400;
        static const int View_SYSTEM_UI_FLAG_IMMERSIVE_STICKY = 0x00001000;
215
        const int SDK_INT = app->activity->sdkVersion;
        if (SDK_INT < 11) {
            return;
        }
220
        GLFMPlatformData *platformData = (GLFMPlatformData *)app->userData;
        JNIEnv *jni = platformData->jniEnv;
        if ((*jni)->ExceptionCheck(jni)) {
            return;
225     }

        jobject decorView = _glfmGetDecorView(app);
        if (!decorView) {
            return;
230     }
        if (uiChrome == GLFMUserInterfaceChromeNavigationAndStatusBar) {
            _glfmCallJavaMethodWithArgs(jni, decorView, "setSystemUiVisibility", "(I↙
                ↳ )V", Void, 0);
        } else if (SDK_INT >= 11 && SDK_INT < 14) {
            _glfmCallJavaMethodWithArgs(jni, decorView, "setSystemUiVisibility", "(I↙
                ↳ )V", Void,
235                                     View_STATUS_BAR_HIDDEN);
        } else if (SDK_INT >= 14 && SDK_INT < 19) {
            if (uiChrome == GLFMUserInterfaceChromeNavigation) {
                _glfmCallJavaMethodWithArgs(jni, decorView, "setSystemUiVisibility",↙
                    ↳ "(I)V", Void,
                                            View_SYSTEM_UI_FLAG_FULLSCREEN);
240             } else {
                _glfmCallJavaMethodWithArgs(jni, decorView, "setSystemUiVisibility",↙
                    ↳ "(I)V", Void,
                                            View_SYSTEM_UI_FLAG_LOW_PROFILE |
                                            View_SYSTEM_UI_FLAG_FULLSCREEN);
            }
245     } else if (SDK_INT >= 19) {
            if (uiChrome == GLFMUserInterfaceChromeNavigation) {
                _glfmCallJavaMethodWithArgs(jni, decorView, "setSystemUiVisibility",↙
                    ↳ "(I)V", Void,
                                            View_SYSTEM_UI_FLAG_FULLSCREEN);
            } else {
250             _glfmCallJavaMethodWithArgs(jni, decorView, "setSystemUiVisibility",↙
                    ↳ "(I)V", Void,
                                            View_SYSTEM_UI_FLAG_HIDE_NAVIGATION |
                                            View_SYSTEM_UI_FLAG_FULLSCREEN |
                                            View_SYSTEM_UI_FLAG_LAYOUT_STABLE |
                                            View_SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION↙
                                                ↳ |
255                                         View_SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN |
                                            View_SYSTEM_UI_FLAG_IMMERSIVE_STICKY);
            }
        }
        (*jni)->DeleteLocalRef(jni, decorView);
```

```
260         _glfmClearJavaException()
    }

    /*
     * Move task to the back if it is root task. This make the back button have the ↙
         ↳ same behavior
265  * as the home button.
     *
     * Without this, when the user presses the back button, the loop in android_main↙
         ↳ () is exited, the
     * OpenGL context is destroyed, and the main thread is destroyed. The ↙
         ↳ android_main() function
     * would be called again in the same process if the user returns to the app.
270  *
     * When this, when the app is in the background, the app will pause in the ↙
         ↳ ALooper_pollAll() call.
     */
    static bool _glfmHandleBackButton(struct android_app *app) {
        GLFMPlatformData *platformData = (GLFMPlatformData *)app->userData;
275     JNIEnv *jni = platformData->jniEnv;
        if ((*jni)->ExceptionCheck(jni)) {
            return false;
        }

280     jboolean handled = _glfmCallJavaMethodWithArgs(jni, app->activity->clazz, "↙
            ↳ moveTaskToBack",
                                                       "(Z)Z", Boolean, false);
        return !_glfmWasJavaExceptionThrown() && handled;
    }

285 static bool _glfmSetKeyboardVisible(GLFMPlatformData *platformData, bool visible↙
        ↳ ) {
        static const int InputMethodManager_SHOW_FORCED = 2;

        JNIEnv *jni = platformData->jniEnv;
        if ((*jni)->ExceptionCheck(jni)) {
290         return false;
        }

        jobject decorView = _glfmGetDecorView(platformData->app);
        if (!decorView) {
295         return false;
        }

        jclass contextClass = (*jni)->FindClass(jni, "android/content/Context");
        if (_glfmWasJavaExceptionThrown()) {
300         return false;
        }

        jstring imString = _glfmGetJavaStaticField(jni, contextClass, "↙
            ↳ INPUT_METHOD_SERVICE",
                                                   "Ljava/lang/String;", Object);
305     if (!imString || _glfmWasJavaExceptionThrown()) {
            return false;
        }
        jobject ime = _glfmCallJavaMethodWithArgs(jni, platformData->app->activity->↙
            ↳ clazz,
```

```
                                                    "getSystemService",
310                                                 "(Ljava/lang/String;)Ljava/lang/↙
                                                        ↳ Object;",
                                                    Object, imString);
        if (!ime || _glfmWasJavaExceptionThrown()) {
            return false;
        }
315
        if (visible) {
            _glfmCallJavaMethodWithArgs(jni, ime, "showSoftInput", "(Landroid/view/↙
                ↳ View;I)Z", Boolean,
                                        decorView, InputMethodManager_SHOW_FORCED);
        } else {
320         jobject windowToken = _glfmCallJavaMethod(jni, decorView, "↙
                ↳ getWindowToken",
                                                      "()Landroid/os/IBinder;", ↙
                                                          ↳ Object);
            if (!windowToken || _glfmWasJavaExceptionThrown()) {
                return false;
            }
325         _glfmCallJavaMethodWithArgs(jni, ime, "hideSoftInputFromWindow",
                                        "(Landroid/os/IBinder;I)Z", Boolean, ↙
                                            ↳ windowToken, 0);
            (*jni)->DeleteLocalRef(jni, windowToken);
        }

330     (*jni)->DeleteLocalRef(jni, ime);
        (*jni)->DeleteLocalRef(jni, imString);
        (*jni)->DeleteLocalRef(jni, contextClass);
        (*jni)->DeleteLocalRef(jni, decorView);

335     return !_glfmWasJavaExceptionThrown();
    }

    static void _glfmResetContentRect(GLFMPlatformData *platformData) {
        // Reset's NativeActivity's content rect so that onContentRectChanged acts ↙
            ↳ as a
340     // OnGlobalLayoutListener. This is needed to detect changes to ↙
            ↳ getWindowVisibleDisplayFrame()
        // HACK: This uses undocumented fields.

        JNIEnv *jni = platformData->jniEnv;
        if ((*jni)->ExceptionCheck(jni)) {
345         return;
        }

        jfieldID field = _glfmGetJavaFieldID(jni, platformData->app->activity->clazz↙
            ↳ ,
                                             "mLastContentWidth", "I");
350     if (!field || _glfmWasJavaExceptionThrown()) {
            return;
        }

        (*jni)->SetIntField(jni, platformData->app->activity->clazz, field, -1);
355     _glfmClearJavaException()
    }
```

```
static ARect _glfmGetWindowVisibleDisplayFrame(GLFMPlatformData *platformData, ↙
    ↳ ARect defaultRect) {
    JNIEnv *jni = platformData->jniEnv;
    if ((*jni)->ExceptionCheck(jni)) {
        return defaultRect;
    }

    jobject decorView = _glfmGetDecorView(platformData->app);
    if (!decorView) {
        return defaultRect;
    }

    jclass javaRectClass = (*jni)->FindClass(jni, "android/graphics/Rect");
    if (_glfmWasJavaExceptionThrown()) {
        return defaultRect;
    }

    jobject javaRect = (*jni)->AllocObject(jni, javaRectClass);
    if (_glfmWasJavaExceptionThrown()) {
        return defaultRect;
    }

    _glfmCallJavaMethodWithArgs(jni, decorView, "getWindowVisibleDisplayFrame",
                                "(Landroid/graphics/Rect;)V", Void, javaRect);
    if (_glfmWasJavaExceptionThrown()) {
        return defaultRect;
    }

    ARect rect;
    rect.left = _glfmGetJavaField(jni, javaRect, "left", "I", Int);
    rect.right = _glfmGetJavaField(jni, javaRect, "right", "I", Int);
    rect.top = _glfmGetJavaField(jni, javaRect, "top", "I", Int);
    rect.bottom = _glfmGetJavaField(jni, javaRect, "bottom", "I", Int);

    (*jni)->DeleteLocalRef(jni, javaRect);
    (*jni)->DeleteLocalRef(jni, javaRectClass);
    (*jni)->DeleteLocalRef(jni, decorView);

    if (_glfmWasJavaExceptionThrown()) {
        return defaultRect;
    } else {
        return rect;
    }
}

static uint32_t _glfmGetUnicodeChar(GLFMPlatformData *platformData, AInputEvent ↙
    ↳ *event) {
    JNIEnv *jni = platformData->jniEnv;
    if ((*jni)->ExceptionCheck(jni)) {
        return 0;
    }

    jint keyCode = AKeyEvent_getKeyCode(event);
    jint metaState = AKeyEvent_getMetaState(event);

    jclass keyEventClass = (*jni)->FindClass(jni, "android/view/KeyEvent");
    if (!keyEventClass || _glfmWasJavaExceptionThrown()) {
```

```
        return 0;
    }

    jmethodID getUnicodeChar = (*jni)->GetMethodID(jni, keyEventClass, "
        ↪ getUnicodeChar", "(I)I");
    jmethodID eventConstructor = (*jni)->GetMethodID(jni, keyEventClass, "<init
        ↪ >", "(II)V");

    jobject eventObject = (*jni)->NewObject(jni, keyEventClass, eventConstructor
        ↪ ,
                                            AKEY_EVENT_ACTION_DOWN, keyCode);
    if (!keyEventClass || _glfmWasJavaExceptionThrown()) {
        return 0;
    }

    jint unicodeKey = (*jni)->CallIntMethod(jni, eventObject, getUnicodeChar, 
        ↪ metaState);

    (*jni)->DeleteLocalRef(jni, eventObject);
    (*jni)->DeleteLocalRef(jni, keyEventClass);

    if (_glfmWasJavaExceptionThrown()) {
        return 0;
    } else {
        return (uint32_t)unicodeKey;
    }
}

// MARK: EGL

static bool _glfmEGLContextInit(GLFMPlatformData *platformData) {

    // Available in eglext.h in API 18
    static const int EGL_CONTEXT_MAJOR_VERSION_KHR = 0x3098;
    static const int EGL_CONTEXT_MINOR_VERSION_KHR = 0x30FB;

    bool created = false;
    if (platformData->eglContext == EGL_NO_CONTEXT) {
        // OpenGL ES 3.2
        if (platformData->display->preferredAPI >= GLFMRenderingAPIOpenGLES32) {
            // TODO: Untested, need an OpenGL ES 3.2 device for testing
            const EGLint contextAttribs[] = {EGL_CONTEXT_MAJOR_VERSION_KHR, 3,
                                             EGL_CONTEXT_MINOR_VERSION_KHR, 2, 
                                                 ↪ EGL_NONE};
            platformData->eglContext = eglCreateContext(platformData->eglDisplay
                ↪ ,
                                                        platformData->eglConfig,
                                                        EGL_NO_CONTEXT, 
                                                            ↪ contextAttribs);
            created = platformData->eglContext != EGL_NO_CONTEXT;
        }
        // OpenGL ES 3.1
        if (!created && platformData->display->preferredAPI >= 
            ↪ GLFMRenderingAPIOpenGLES31) {
            // TODO: Untested, need an OpenGL ES 3.1 device for testing
            const EGLint contextAttribs[] = {EGL_CONTEXT_MAJOR_VERSION_KHR, 3,
                                             EGL_CONTEXT_MINOR_VERSION_KHR, 1, 
```

```
                                                    ↳ EGL_NONE};
            platformData->eglContext = eglCreateContext(platformData->eglDisplay ↙
                ↳ ,
                                                        platformData->eglConfig ,
                                                        EGL_NO_CONTEXT, ↙
                                                            ↳ contextAttribs);
            created = platformData->eglContext != EGL_NO_CONTEXT;
        }
        // OpenGL ES 3.0
        if (!created && platformData->display->preferredAPI >= ↙
            ↳ GLFMRenderingAPIOpenGLES3) {
            const EGLint contextAttribs[] = {EGL_CONTEXT_CLIENT_VERSION, 3, ↙
                ↳ EGL_NONE, EGL_NONE};
            platformData->eglContext = eglCreateContext(platformData->eglDisplay ↙
                ↳ ,
                                                        platformData->eglConfig ,
                                                        EGL_NO_CONTEXT, ↙
                                                            ↳ contextAttribs);
            created = platformData->eglContext != EGL_NO_CONTEXT;
        }
        // OpenGL ES 2.0
        if (!created) {
            const EGLint contextAttribs[] = {EGL_CONTEXT_CLIENT_VERSION, 2, ↙
                ↳ EGL_NONE, EGL_NONE};
            platformData->eglContext = eglCreateContext(platformData->eglDisplay ↙
                ↳ ,
                                                        platformData->eglConfig ,
                                                        EGL_NO_CONTEXT, ↙
                                                            ↳ contextAttribs);
            created = platformData->eglContext != EGL_NO_CONTEXT;
        }

        if (created) {
            EGLint majorVersion = 0;
            EGLint minorVersion = 0;
            eglQueryContext(platformData->eglDisplay, platformData->eglContext,
                            EGL_CONTEXT_MAJOR_VERSION_KHR, &majorVersion);
            if (majorVersion >= 3) {
                eglQueryContext(platformData->eglDisplay, platformData->↙
                    ↳ eglContext,
                                EGL_CONTEXT_MINOR_VERSION_KHR, &minorVersion);
            }
            if (majorVersion == 3 && minorVersion == 1) {
                platformData->renderingAPI = GLFMRenderingAPIOpenGLES31;
            } else if (majorVersion == 3) {
                platformData->renderingAPI = GLFMRenderingAPIOpenGLES3;
            } else {
                platformData->renderingAPI = GLFMRenderingAPIOpenGLES2;
            }
        }
    }

    if (!eglMakeCurrent(platformData->eglDisplay, platformData->eglSurface,
                        platformData->eglSurface, platformData->eglContext)) {
        LOG_LIFECYCLE("eglMakeCurrent() failed");
        platformData->eglContextCurrent = false;
        return false;
```

```
    } else {
        platformData->eglContextCurrent = true;
        if (created && platformData->display) {
            LOG_LIFECYCLE("GL Context made current");
            if (platformData->display->surfaceCreatedFunc) {
                platformData->display->surfaceCreatedFunc(platformData->display,
                                                          platformData->width,
                                                          platformData->height);
            }
        }
        return true;
    }
}

static void _glfmEGLContextDisable(GLFMPlatformData *platformData) {
    if (platformData->eglDisplay != EGL_NO_DISPLAY) {
        eglMakeCurrent(platformData->eglDisplay, EGL_NO_SURFACE, EGL_NO_SURFACE,
            EGL_NO_CONTEXT);
    }
    platformData->eglContextCurrent = false;
}

static void _glfmEGLSurfaceInit(GLFMPlatformData *platformData) {
    if (platformData->eglSurface == EGL_NO_SURFACE) {
        platformData->eglSurface = eglCreateWindowSurface(platformData->
            eglDisplay,
                                                          platformData->
                                                              eglConfig,
                                                          platformData->app->
                                                              window, NULL);
    }
}

static void _glfmEGLLogConfig(GLFMPlatformData *platformData, EGLConfig config) {
    LOG_DEBUG("Config: %p", config);
    EGLint value;
    eglGetConfigAttrib(platformData->eglDisplay, config, EGL_RENDERABLE_TYPE, &
        value);
    LOG_DEBUG("  EGL_RENDERABLE_TYPE %i", value);
    eglGetConfigAttrib(platformData->eglDisplay, config, EGL_SURFACE_TYPE, &
        value);
    LOG_DEBUG("  EGL_SURFACE_TYPE    %i", value);
    eglGetConfigAttrib(platformData->eglDisplay, config, EGL_RED_SIZE, &value);
    LOG_DEBUG("  EGL_RED_SIZE        %i", value);
    eglGetConfigAttrib(platformData->eglDisplay, config, EGL_GREEN_SIZE, &value)
        ;
    LOG_DEBUG("  EGL_GREEN_SIZE      %i", value);
    eglGetConfigAttrib(platformData->eglDisplay, config, EGL_BLUE_SIZE, &value);
    LOG_DEBUG("  EGL_BLUE_SIZE       %i", value);
    eglGetConfigAttrib(platformData->eglDisplay, config, EGL_ALPHA_SIZE, &value)
        ;
    LOG_DEBUG("  EGL_ALPHA_SIZE      %i", value);
    eglGetConfigAttrib(platformData->eglDisplay, config, EGL_DEPTH_SIZE, &value)
        ;
    LOG_DEBUG("  EGL_DEPTH_SIZE      %i", value);
    eglGetConfigAttrib(platformData->eglDisplay, config, EGL_STENCIL_SIZE, &
```

```
        ↳ value);
    LOG_DEBUG("  EGL_STENCIL_SIZE     %i", value);
    eglGetConfigAttrib(platformData->eglDisplay, config, EGL_SAMPLE_BUFFERS, &↙
        ↳ value);
    LOG_DEBUG("  EGL_SAMPLE_BUFFERS  %i", value);
    eglGetConfigAttrib(platformData->eglDisplay, config, EGL_SAMPLES, &value);
    LOG_DEBUG("  EGL_SAMPLES         %i", value);
}

static bool _glfmEGLInit(GLFMPlatformData *platformData) {
    if (platformData->eglDisplay != EGL_NO_DISPLAY) {
        _glfmEGLSurfaceInit(platformData);
        return _glfmEGLContextInit(platformData);
    }
    int rBits, gBits, bBits, aBits;
    int depthBits, stencilBits, samples;

    switch (platformData->display->colorFormat) {
        case GLFMColorFormatRGB565:
            rBits = 5;
            gBits = 6;
            bBits = 5;
            aBits = 0;
            break;
        case GLFMColorFormatRGBA8888:
        default:
            rBits = 8;
            gBits = 8;
            bBits = 8;
            aBits = 8;
            break;
    }

    switch (platformData->display->depthFormat) {
        case GLFMDepthFormatNone:
        default:
            depthBits = 0;
            break;
        case GLFMDepthFormat16:
            depthBits = 16;
            break;
        case GLFMDepthFormat24:
            depthBits = 24;
            break;
    }

    switch (platformData->display->stencilFormat) {
        case GLFMStencilFormatNone:
        default:
            stencilBits = 0;
            break;
        case GLFMStencilFormat8:
            stencilBits = 8;
            if (depthBits > 0) {
                // Many implementations only allow 24-bit depth with 8-bit ↙
                    ↳ stencil.
                depthBits = 24;
```

```
            }
            break;
        }

        samples = platformData->display->multisample == GLFMMultisample4X ? 4 : 0;

        EGLint majorVersion;
        EGLint minorVersion;
        EGLint format;
        EGLint numConfigs;

        platformData->eglDisplay = eglGetDisplay(EGL_DEFAULT_DISPLAY);
        eglInitialize(platformData->eglDisplay, &majorVersion, &minorVersion);

        while (true) {
            const EGLint attribs[] = {
                EGL_RENDERABLE_TYPE, EGL_OPENGL_ES2_BIT,
                EGL_SURFACE_TYPE, EGL_WINDOW_BIT,
                EGL_RED_SIZE, rBits,
                EGL_GREEN_SIZE, gBits,
                EGL_BLUE_SIZE, bBits,
                EGL_ALPHA_SIZE, aBits,
                EGL_DEPTH_SIZE, depthBits,
                EGL_STENCIL_SIZE, stencilBits,
                EGL_SAMPLE_BUFFERS, samples > 0 ? 1 : 0,
                EGL_SAMPLES, samples > 0 ? samples : 0,
                EGL_NONE};

            eglChooseConfig(platformData->eglDisplay, attribs, &platformData->↙
                ↳ eglConfig, 1, &numConfigs);
            if (numConfigs) {
                // Found!
                //_glfmEGLLogConfig(platformData, platformData->eglConfig);
                break;
            } else if (samples > 0) {
                // Try 2x multisampling or no multisampling
                samples -= 2;
            } else if (depthBits > 8) {
                // Try 16-bit depth or 8-bit depth
                depthBits -= 8;
            } else {
                // Failure
                static bool printedConfigs = false;
                if (!printedConfigs) {
                    printedConfigs = true;
                    LOG_DEBUG("eglChooseConfig() failed");
                    EGLConfig configs[256];
                    EGLint numTotalConfigs;
                    if (eglGetConfigs(platformData->eglDisplay, configs, 256, &↙
                        ↳ numTotalConfigs)) {
                        LOG_DEBUG("Num available configs: %i", numTotalConfigs);
                        int i;
                        for (i = 0; i < numTotalConfigs; i++) {
                            _glfmEGLLogConfig(platformData, configs[i]);
                        }
                    } else {
                        LOG_DEBUG("Couldn't get any EGL configs");
```

```
                }
            }

            _glfmReportSurfaceError(platformData->eglDisplay, "eglChooseConfig() failed");
            eglTerminate(platformData->eglDisplay);
            platformData->eglDisplay = EGL_NO_DISPLAY;
            return false;
        }
    }

    _glfmEGLSurfaceInit(platformData);

    eglQuerySurface(platformData->eglDisplay, platformData->eglSurface, EGL_WIDTH,
                    &platformData->width);
    eglQuerySurface(platformData->eglDisplay, platformData->eglSurface, EGL_HEIGHT,
                    &platformData->height);
    eglGetConfigAttrib(platformData->eglDisplay, platformData->eglConfig, EGL_NATIVE_VISUAL_ID,
                       &format);

    ANativeWindow_setBuffersGeometry(platformData->app->window, 0, 0, format);

    return _glfmEGLContextInit(platformData);
}

static void _glfmEGLSurfaceDestroy(GLFMPlatformData *platformData) {
    if (platformData->eglSurface != EGL_NO_SURFACE) {
        eglDestroySurface(platformData->eglDisplay, platformData->eglSurface);
        platformData->eglSurface = EGL_NO_SURFACE;
    }
    _glfmEGLContextDisable(platformData);
}

static void _glfmEGLDestroy(GLFMPlatformData *platformData) {
    if (platformData->eglDisplay != EGL_NO_DISPLAY) {
        eglMakeCurrent(platformData->eglDisplay, EGL_NO_SURFACE, EGL_NO_SURFACE, EGL_NO_CONTEXT);
        if (platformData->eglContext != EGL_NO_CONTEXT) {
            eglDestroyContext(platformData->eglDisplay, platformData->eglContext);
            if (platformData->display) {
                LOG_LIFECYCLE("GL Context destroyed");
                if (platformData->display->surfaceDestroyedFunc) {
                    platformData->display->surfaceDestroyedFunc(platformData->display);
                }
            }
        }
        if (platformData->eglSurface != EGL_NO_SURFACE) {
            eglDestroySurface(platformData->eglDisplay, platformData->eglSurface);
        }
        eglTerminate(platformData->eglDisplay);
    }
```

```
    platformData->eglDisplay = EGL_NO_DISPLAY;
    platformData->eglContext = EGL_NO_CONTEXT;
    platformData->eglSurface = EGL_NO_SURFACE;
    platformData->eglContextCurrent = false;
}

static void _glfmEGLCheckError(GLFMPlatformData *platformData) {
    EGLint err = eglGetError();
    if (err == EGL_BAD_SURFACE) {
        _glfmEGLSurfaceDestroy(platformData);
        _glfmEGLSurfaceInit(platformData);
    } else if (err == EGL_CONTEXT_LOST || err == EGL_BAD_CONTEXT) {
        if (platformData->eglContext != EGL_NO_CONTEXT) {
            platformData->eglContext = EGL_NO_CONTEXT;
            platformData->eglContextCurrent = false;
            if (platformData->display) {
                LOG_LIFECYCLE("GL Context lost");
                if (platformData->display->surfaceDestroyedFunc) {
                    platformData->display->surfaceDestroyedFunc(platformData->display);
                }
            }
        }
        _glfmEGLContextInit(platformData);
    } else {
        _glfmEGLDestroy(platformData);
        _glfmEGLInit(platformData);
    }
}

static void _glfmDrawFrame(GLFMPlatformData *platformData) {
    if (!platformData->eglContextCurrent) {
        // Probably a bad config (Happens on Android 2.3 emulator)
        return;
    }

    // Check for resize (or rotate)
    int32_t width;
    int32_t height;
    eglQuerySurface(platformData->eglDisplay, platformData->eglSurface, EGL_WIDTH, &width);
    eglQuerySurface(platformData->eglDisplay, platformData->eglSurface, EGL_HEIGHT, &height);
    if (width != platformData->width || height != platformData->height) {
        LOG_LIFECYCLE("Resize: %i x %i", width, height);
        platformData->width = width;
        platformData->height = height;
        if (platformData->display && platformData->display->surfaceResizedFunc) {
            platformData->display->surfaceResizedFunc(platformData->display, width, height);
        }
    }

    // Tick and draw
    if (platformData->display && platformData->display->mainLoopFunc) {
        const double frameTime = _glfmTimeSeconds(
```

```
765                     _glfmTimeSubstract(_glfmTimeNow(), platformData->initTime));
            platformData->display->mainLoopFunc(platformData->display, frameTime);
        } else {
            glClearColor(0.0, 0.0, 0.0, 1.0);
            glClear(GL_COLOR_BUFFER_BIT);
770     }

        // Swap
        if (!eglSwapBuffers(platformData->eglDisplay, platformData->eglSurface)) {
            _glfmEGLCheckError(platformData);
775     }
    }

    // MARK: Native app glue extension

780 static bool ARectsEqual(ARect r1, ARect r2) {
        return r1.left == r2.left && r1.top == r2.top && r1.right == r2.right && r1.↙
            ↳ bottom == r2.bottom;
    }

    static void _glfmWriteCmd(struct android_app *android_app, int8_t cmd) {
785     write(android_app->msgwrite, &cmd, sizeof(cmd));
    }

    static void _glfmSetContentRect(struct android_app *android_app, ARect rect) {
        pthread_mutex_lock(&android_app->mutex);
790     android_app->pendingContentRect = rect;
        _glfmWriteCmd(android_app, APP_CMD_CONTENT_RECT_CHANGED);
        while (!ARectsEqual(android_app->contentRect, android_app->↙
            ↳ pendingContentRect)) {
            pthread_cond_wait(&android_app->cond, &android_app->mutex);
        }
795     pthread_mutex_unlock(&android_app->mutex);
    }

    static void _glfmOnContentRectChanged(ANativeActivity *activity, const ARect *↙
        ↳ rect) {
        _glfmSetContentRect((struct android_app *)activity->instance, *rect);
800 }

    // MARK: Keyboard visibility

    static void _glfmUpdateKeyboardVisibility(GLFMPlatformData *platformData) {
805     if (platformData->display) {
            ARect windowRect = platformData->app->contentRect;
            ARect visibleRect = _glfmGetWindowVisibleDisplayFrame(platformData, ↙
                ↳ windowRect);
            ARect nonVisibleRect[4];

810         // Left
            nonVisibleRect[0].left = windowRect.left;
            nonVisibleRect[0].right = visibleRect.left;
            nonVisibleRect[0].top = windowRect.top;
            nonVisibleRect[0].bottom = windowRect.bottom;
815
            // Right
            nonVisibleRect[1].left = visibleRect.right;
```

```
            nonVisibleRect[1].right = windowRect.right;
            nonVisibleRect[1].top = windowRect.top;
820         nonVisibleRect[1].bottom = windowRect.bottom;

            // Top
            nonVisibleRect[2].left = windowRect.left;
            nonVisibleRect[2].right = windowRect.right;
825         nonVisibleRect[2].top = windowRect.top;
            nonVisibleRect[2].bottom = visibleRect.top;

            // Bottom
            nonVisibleRect[3].left = windowRect.left;
830         nonVisibleRect[3].right = windowRect.right;
            nonVisibleRect[3].top = visibleRect.bottom;
            nonVisibleRect[3].bottom = windowRect.bottom;

            // Find largest with minimum keyboard size
835         const int minimumKeyboardSize = (int)(100 * platformData->scale);
            int largestIndex = 0;
            int largestArea = -1;
            for (int i = 0; i < 4; i++) {
                int w = nonVisibleRect[i].right - nonVisibleRect[i].left;
840             int h = nonVisibleRect[i].bottom - nonVisibleRect[i].top;
                int area = w * h;
                if (w >= minimumKeyboardSize && h >= minimumKeyboardSize && area > ↙
                    ↳ largestArea) {
                    largestIndex = i;
                    largestArea = area;
845             }
            }

            bool keyboardVisible = largestArea > 0;
            ARect keyboardFrame = keyboardVisible ? nonVisibleRect[largestIndex] : (↙
                ↳ ARect){0, 0, 0, 0};
850
            // Send update notification
            if (platformData->keyboardVisible != keyboardVisible ||
                    !ARectsEqual(platformData->keyboardFrame, keyboardFrame)) {
                platformData->keyboardVisible = keyboardVisible;
855             platformData->keyboardFrame = keyboardFrame;
                if (platformData->display->keyboardVisibilityChangedFunc) {
                    double x = keyboardFrame.left;
                    double y = keyboardFrame.top;
                    double w = keyboardFrame.right - keyboardFrame.left;
860                 double h = keyboardFrame.bottom - keyboardFrame.top;
                    platformData->display->keyboardVisibilityChangedFunc(↙
                        ↳ platformData->display,
                                                                         keyboardVisible↙
                                                                             ↳ ,
                                                                         x, y, w, h)↙
                                                                             ↳ ;
                }
865         }
        }
    }

    // MARK: App command callback
```

```
870
static void _glfmSetAnimating(GLFMPlatformData *platformData, bool animating) {
    if (platformData->animating != animating) {
        bool sendAppEvent = true;
        if (!platformData->hasInited && animating) {
875         platformData->hasInited = true;
            platformData->initTime = _glfmTimeNow();
            sendAppEvent = false;
        }
        platformData->animating = animating;
880     if (sendAppEvent && platformData->display && platformData->display->↙
            ↳ focusFunc) {
            platformData->display->focusFunc(platformData->display, animating);
        }
    }
}
885
static void _glfmOnAppCmd(struct android_app *app, int32_t cmd) {
    GLFMPlatformData *platformData = (GLFMPlatformData *)app->userData;
    switch (cmd) {
        case APP_CMD_SAVE_STATE: {
890         LOG_LIFECYCLE("APP_CMD_SAVE_STATE");
            break;
        }
        case APP_CMD_INIT_WINDOW: {
            LOG_LIFECYCLE("APP_CMD_INIT_WINDOW");
895         const bool success = _glfmEGLInit(platformData);
            if (!success) {
                _glfmEGLCheckError(platformData);
            }
            _glfmDrawFrame(platformData);
900         break;
        }
        case APP_CMD_WINDOW_RESIZED: {
            LOG_LIFECYCLE("APP_CMD_WINDOW_RESIZED");
            break;
905     }
        case APP_CMD_TERM_WINDOW: {
            LOG_LIFECYCLE("APP_CMD_TERM_WINDOW");
            _glfmEGLSurfaceDestroy(platformData);
            _glfmSetAnimating(platformData, false);
910         break;
        }
        case APP_CMD_WINDOW_REDRAW_NEEDED: {
            LOG_LIFECYCLE("APP_CMD_WINDOW_REDRAW_NEEDED");
            _glfmDrawFrame(platformData);
915         break;
        }
        case APP_CMD_GAINED_FOCUS: {
            LOG_LIFECYCLE("APP_CMD_GAINED_FOCUS");
            _glfmSetAnimating(platformData, true);
920         break;
        }
        case APP_CMD_LOST_FOCUS: {
            LOG_LIFECYCLE("APP_CMD_LOST_FOCUS");
            if (platformData->animating) {
925             _glfmDrawFrame(platformData);
```

```
                _glfmSetAnimating(platformData, false);
            }
            break;
        }
        case APP_CMD_CONTENT_RECT_CHANGED: {
            LOG_LIFECYCLE("APP_CMD_CONTENT_RECT_CHANGED");
            pthread_mutex_lock(&app->mutex);
            app->contentRect = app->pendingContentRect;
            _glfmResetContentRect(platformData);
            pthread_cond_broadcast(&app->cond);
            pthread_mutex_unlock(&app->mutex);
            _glfmUpdateKeyboardVisibility(platformData);
            break;
        }
        case APP_CMD_LOW_MEMORY: {
            LOG_LIFECYCLE("APP_CMD_LOW_MEMORY");
            if (platformData->display && platformData->display->lowMemoryFunc) {
                platformData->display->lowMemoryFunc(platformData->display);
            }
            break;
        }
        case APP_CMD_START: {
            LOG_LIFECYCLE("APP_CMD_START");
            _glfmSetFullScreen(app, platformData->display->uiChrome);
            break;
        }
        case APP_CMD_RESUME: {
            LOG_LIFECYCLE("APP_CMD_RESUME");
            break;
        }
        case APP_CMD_PAUSE: {
            LOG_LIFECYCLE("APP_CMD_PAUSE");
            break;
        }
        case APP_CMD_STOP: {
            LOG_LIFECYCLE("APP_CMD_STOP");
            break;
        }
        case APP_CMD_DESTROY: {
            LOG_LIFECYCLE("APP_CMD_DESTROY");
            _glfmEGLDestroy(platformData);
            break;
        }
        default: {
            // Do nothing
            break;
        }
    }
}

// MARK: Key and touch input callback

static const char *_glfmUnicodeToUTF8(uint32_t unicode) {
    static char utf8[5];
    if (unicode < 0x80) {
        utf8[0] = (char)(unicode & 0x7f);
        utf8[1] = 0;
```

```
    } else if (unicode < 0x800) {
        utf8[0] = (char)(0xc0 | (unicode >> 6));
        utf8[1] = (char)(0x80 | (unicode & 0x3f));
        utf8[2] = 0;
    } else if (unicode < 0x10000) {
        utf8[0] = (char)(0xe0 | (unicode >> 12));
        utf8[1] = (char)(0x80 | ((unicode >> 6) & 0x3f));
        utf8[2] = (char)(0x80 | (unicode & 0x3f));
        utf8[3] = 0;
    } else if (unicode < 0x110000) {
        utf8[0] = (char)(0xf0 | (unicode >> 18));
        utf8[1] = (char)(0x80 | ((unicode >> 12) & 0x3f));
        utf8[2] = (char)(0x80 | ((unicode >> 6) & 0x3f));
        utf8[3] = (char)(0x80 | (unicode & 0x3f));
        utf8[4] = 0;
    } else {
        utf8[0] = 0;
    }
    return utf8;
}

static int32_t _glfmOnInputEvent(struct android_app *app, AInputEvent *event) {
    GLFMPlatformData *platformData = (GLFMPlatformData *)app->userData;
    const int32_t eventType = AInputEvent_getType(event);
    if (eventType == AINPUT_EVENT_TYPE_KEY) {
        int handled = 0;
        if (platformData->display && platformData->display->keyFunc) {
            int32_t aKeyCode = AKeyEvent_getKeyCode(event);
            int32_t aAction = AKeyEvent_getAction(event);
            if (aKeyCode != 0) {
                GLFMKey key;
                switch (aKeyCode) {
                    case AKEYCODE_DPAD_LEFT:
                        key = GLFMKeyLeft;
                        break;
                    case AKEYCODE_DPAD_RIGHT:
                        key = GLFMKeyRight;
                        break;
                    case AKEYCODE_DPAD_UP:
                        key = GLFMKeyUp;
                        break;
                    case AKEYCODE_DPAD_DOWN:
                        key = GLFMKeyDown;
                        break;
                    case AKEYCODE_ENTER:
                    case AKEYCODE_DPAD_CENTER:
                        key = GLFMKeyEnter;
                        break;
                    case AKEYCODE_TAB:
                        key = GLFMKeyTab;
                        break;
                    case AKEYCODE_SPACE:
                        key = GLFMKeySpace;
                        break;
                    case AKEYCODE_BACK:
                        key = GLFMKeyNavBack;
                        break;
```

```
1040                        case AKEYCODE_MENU:
                                key = GLFMKeyNavMenu;
                                break;
                            default:
                                // TODO: Send all keycodes?
1045                            if (aKeyCode >= AKEYCODE_0 && aKeyCode <= AKEYCODE_9) {
                                    key = (GLFMKey)(aKeyCode - AKEYCODE_0 + '0');
                                } else if (aKeyCode >= AKEYCODE_A && aKeyCode <= ↙
                                    ↳ AKEYCODE_Z) {
                                    key = (GLFMKey)(aKeyCode - AKEYCODE_A + 'A');
                                } else {
1050                                key = (GLFMKey)0;
                                }
                                break;
                        }

1055                    if (key != 0) {
                            if (aAction == AKEY_EVENT_ACTION_UP) {
                                handled = platformData->display->keyFunc(platformData->↙
                                    ↳ display, key,
                                                                         GLFMKeyActionReleased↙
                                                                             ↳ , 0);
                                if (handled == 0 && aKeyCode == AKEYCODE_BACK) {
1060                                handled = _glfmHandleBackButton(app) ? 1 : 0;
                                }
                            } else if (aAction == AKEY_EVENT_ACTION_DOWN) {
                                GLFMKeyAction keyAction;
                                if (AKeyEvent_getRepeatCount(event) > 0) {
1065                                keyAction = GLFMKeyActionRepeated;
                                } else {
                                    keyAction = GLFMKeyActionPressed;
                                }
                                handled = platformData->display->keyFunc(platformData->↙
                                    ↳ display, key,
1070                                                                     keyAction, 0);
                            } else if (aAction == AKEY_EVENT_ACTION_MULTIPLE) {
                                int32_t i;
                                for (i = AKeyEvent_getRepeatCount(event); i > 0; i--) {
                                    handled |= platformData->display->keyFunc(↙
                                        ↳ platformData->display, key,
1075                                                                     GLFMKeyActionPressed↙
                                                                             ↳ , 0);
                                    handled |= platformData->display->keyFunc(↙
                                        ↳ platformData->display, key,
                                                                         GLFMKeyActionReleased↙
                                                                             ↳ , 0);
                                }
                            }
1080                    }
                    }
                }
                if (platformData->display && platformData->display->charFunc) {
                    int32_t aAction = AKeyEvent_getAction(event);
1085                if (aAction == AKEY_EVENT_ACTION_DOWN || aAction == ↙
                        ↳ AKEY_EVENT_ACTION_MULTIPLE) {
                        uint32_t unicode = _glfmGetUnicodeChar(platformData, event);
                        if (unicode >= ' ') {
```

```
                        const char *str = _glfmUnicodeToUTF8(unicode);
                        if (aAction == AKEY_EVENT_ACTION_DOWN) {
1090                        platformData->display->charFunc(platformData->display, ↙
                                ↳ str, 0);
                        } else {
                            int32_t i;
                            for (i = AKeyEvent_getRepeatCount(event); i > 0; i--) {
                                platformData->display->charFunc(platformData->↙
                                    ↳ display, str, 0);
1095                        }
                        }
                    }
                }
            }
1100        return handled;
        } else if (eventType == AINPUT_EVENT_TYPE_MOTION) {
            if (platformData->display && platformData->display->touchFunc) {
                const int maxTouches = platformData->multitouchEnabled ? ↙
                    ↳ MAX_SIMULTANEOUS_TOUCHES : 1;
                const int32_t action = AMotionEvent_getAction(event);
1105            const int maskedAction = action & AMOTION_EVENT_ACTION_MASK;

                GLFMTouchPhase phase;
                bool validAction = true;

1110            switch (maskedAction) {
                    case AMOTION_EVENT_ACTION_DOWN:
                    case AMOTION_EVENT_ACTION_POINTER_DOWN:
                        phase = GLFMTouchPhaseBegan;
                        break;
1115                case AMOTION_EVENT_ACTION_UP:
                    case AMOTION_EVENT_ACTION_POINTER_UP:
                    case AMOTION_EVENT_ACTION_OUTSIDE:
                        phase = GLFMTouchPhaseEnded;
                        break;
1120                case AMOTION_EVENT_ACTION_MOVE:
                        phase = GLFMTouchPhaseMoved;
                        break;
                    case AMOTION_EVENT_ACTION_CANCEL:
                        phase = GLFMTouchPhaseCancelled;
1125                    break;
                    default:
                        phase = GLFMTouchPhaseCancelled;
                        validAction = false;
                        break;
1130            }
                if (validAction) {
                    if (phase == GLFMTouchPhaseMoved) {
                        const size_t count = AMotionEvent_getPointerCount(event);
                        size_t i;
1135                    for (i = 0; i < count; i++) {
                            const int touchNumber = AMotionEvent_getPointerId(event,↙
                                ↳ i);
                            if (touchNumber >= 0 && touchNumber < maxTouches) {
                                double x = (double)AMotionEvent_getX(event, i);
                                double y = (double)AMotionEvent_getY(event, i);
1140                            platformData->display->touchFunc(platformData->↙
```

```
                                    ↳ display, touchNumber,
                                                                    phase, x, y);
                                //LOG_DEBUG("Touch %i: (%i) %i,%i", touchNumber, ↙
                                    ↳ phase, x, y);
                            }
                        }
1145                } else {
                        const size_t index =
                            (size_t)((action & ↙
                                ↳ AMOTION_EVENT_ACTION_POINTER_INDEX_MASK) >>
                                    AMOTION_EVENT_ACTION_POINTER_INDEX_SHIFT);
                        const int touchNumber = AMotionEvent_getPointerId(event, ↙
                            ↳ index);
1150                    if (touchNumber >= 0 && touchNumber < maxTouches) {
                            double x = (double)AMotionEvent_getX(event, index);
                            double y = (double)AMotionEvent_getY(event, index);
                            platformData->display->touchFunc(platformData->display, ↙
                                ↳ touchNumber,
                                                                    phase, x, y);
1155                        //LOG_DEBUG("Touch %i: (%i) %i,%i", touchNumber, phase, ↙
                                ↳ x, y);
                        }
                    }
                }
            }
1160        return 1;
        }
        return 0;
    }

1165 // MARK: Main entry point

    void android_main(struct android_app *app) {
    #pragma clang diagnostic push
    #pragma clang diagnostic ignored "-Wdeprecated-declarations"
1170    // Don't strip glue code. Although this is deprecated, it's easier with ↙
            ↳ complex CMake files.
        app_dummy();
    #pragma clang diagnostic pop

        LOG_LIFECYCLE("android_main");
1175
        // Init platform data
        GLFMPlatformData *platformData;
        if (platformDataGlobal == NULL) {
            platformDataGlobal = calloc(1, sizeof(GLFMPlatformData));
1180    }
        platformData = platformDataGlobal;

        app->userData = platformData;
        app->onAppCmd = _glfmOnAppCmd;
1185    app->onInputEvent = _glfmOnInputEvent;
        app->activity->callbacks->onContentRectChanged = _glfmOnContentRectChanged;
        platformData->app = app;

        // Init java env
1190    JavaVM *vm = app->activity->vm;
```

```
    (*vm)->AttachCurrentThread(vm, &platformData->jniEnv, NULL);

    // Get display scale
    const int ACONFIGURATION_DENSITY_ANY = 0xfffe; // Added in API 21
1195    const int32_t density = AConfiguration_getDensity(app->config);
    if (density == ACONFIGURATION_DENSITY_DEFAULT || density ==
        ↪ ACONFIGURATION_DENSITY_NONE ||
            density == ACONFIGURATION_DENSITY_ANY || density <= 0) {
        platformData->scale = 1.0;
    } else {
1200        platformData->scale = density / 160.0;
    }

    if (platformData->display == NULL) {
        LOG_LIFECYCLE("glfmMain");
1205        // Only call glfmMain() once per instance
        // This should call glfmInit()
        platformData->display = calloc(1, sizeof(GLFMDisplay));
        platformData->display->platformData = platformData;
        glfmMain(platformData->display);
1210    }

    // Setup window params
    int32_t windowFormat;
    switch (platformData->display->colorFormat) {
1215        case GLFMColorFormatRGB565:
            windowFormat = WINDOW_FORMAT_RGB_565;
            break;
        case GLFMColorFormatRGBA8888: default:
            windowFormat = WINDOW_FORMAT_RGBA_8888;
1220            break;
    }
    bool fullscreen = platformData->display->uiChrome ==
        ↪ GLFMUserInterfaceChromeFullscreen;
    ANativeActivity_setWindowFormat(app->activity, windowFormat);
    ANativeActivity_setWindowFlags(app->activity,
1225                                   fullscreen ? AWINDOW_FLAG_FULLSCREEN : 0,
                                   AWINDOW_FLAG_FULLSCREEN);
    _glfmSetFullScreen(app, platformData->display->uiChrome);

    // Run the main loop
1230    while (1) {
        int ident;
        int events;
        struct android_poll_source *source;

1235        while ((ident = ALooper_pollAll(platformData->animating ? 0 : -1, NULL,
            ↪ &events,
                                        (void **)&source)) >= 0) {
            if (source) {
                source->process(app, source);
            }
1240
//          if (ident == LOOPER_ID_USER) {
//              if (platformData->accelerometerSensor != NULL) {
//                  ASensorEvent event;
//                  while (ASensorEventQueue_getEvents(platformData->
```

```
        ↪ sensorEventQueue,
1245    //                                                  &event, 1) > 0) {
        //                         LOG_DEBUG("accelerometer: x=%f y=%f z=%f",
        //                                 event.acceleration.x, event.acceleration.y,
        //                                 event.acceleration.z);
        //                     }
1250    //                 }
        //             }

                    if (app->destroyRequested != 0) {
                        LOG_LIFECYCLE("Destroying thread");
1255                    _glfmEGLDestroy(platformData);
                        _glfmSetAnimating(platformData, false);
                        (*vm)->DetachCurrentThread(vm);
                        platformData->app = NULL;
                        // App is destroyed, but android_main() can be called again in ↙
                            ↪ the same process.
1260                    return;
                    }
                }

                if (platformData->animating && platformData->display) {
1265                _glfmDrawFrame(platformData);
                }
            }
        }

1270    // MARK: GLFM implementation

        void glfmSetUserInterfaceOrientation(GLFMDisplay *display,
                                             GLFMUserInterfaceOrientation ↙
                                                 ↪ allowedOrientations) {
            if (display->allowedOrientations != allowedOrientations) {
1275            display->allowedOrientations = allowedOrientations;
                GLFMPlatformData *platformData = (GLFMPlatformData *)display->↙
                    ↪ platformData;
                _glfmSetOrientation(platformData->app);
            }
        }
1280
        void glfmGetDisplaySize(GLFMDisplay *display, int *width, int *height) {
            GLFMPlatformData *platformData = (GLFMPlatformData *)display->platformData;
            *width = platformData->width;
            *height = platformData->height;
1285    }

        double glfmGetDisplayScale(GLFMDisplay *display) {
            GLFMPlatformData *platformData = (GLFMPlatformData *)display->platformData;
            return platformData->scale;
1290    }

        void glfmGetDisplayChromeInsets(GLFMDisplay *display, double *top, double *right↙
            ↪ , double *bottom,
                                        double *left) {
            GLFMPlatformData *platformData = (GLFMPlatformData *)display->platformData;
1295        ARect windowRect = platformData->app->contentRect;
            ARect visibleRect = _glfmGetWindowVisibleDisplayFrame(platformData, ↙
```

```
        ↪ windowRect);
    if (visibleRect.right - visibleRect.left <= 0 || visibleRect.bottom - ↙
        ↪ visibleRect.top <= 0) {
        *top = 0;
        *right = 0;
        *bottom = 0;
        *left = 0;
    } else {
        *top = visibleRect.top;
        *right = platformData->width - visibleRect.right;
        *bottom = platformData->height - visibleRect.bottom;
        *left = visibleRect.left;
    }
}

void _glfmDisplayChromeUpdated(GLFMDisplay *display) {
    GLFMPlatformData *platformData = (GLFMPlatformData *)display->platformData;
    _glfmSetFullScreen(platformData->app, display->uiChrome);
}

GLFMRenderingAPI glfmGetRenderingAPI(GLFMDisplay *display) {
    GLFMPlatformData *platformData = (GLFMPlatformData *)display->platformData;
    return platformData->renderingAPI;
}

bool glfmHasTouch(GLFMDisplay *display) {
    (void)display;
    // This will need to change, for say, TV apps
    return true;
}

void glfmSetMouseCursor(GLFMDisplay *display, GLFMMouseCursor mouseCursor) {
    (void)display;
    (void)mouseCursor;
    // Do nothing
}

void glfmSetMultitouchEnabled(GLFMDisplay *display, bool multitouchEnabled) {
    GLFMPlatformData *platformData = (GLFMPlatformData *)display->platformData;
    platformData->multitouchEnabled = multitouchEnabled;
}

bool glfmGetMultitouchEnabled(GLFMDisplay *display) {
    GLFMPlatformData *platformData = (GLFMPlatformData *)display->platformData;
    return platformData->multitouchEnabled;
}

GLFMProc glfmGetProcAddress(const char *functionName) {
    GLFMProc function = eglGetProcAddress(functionName);
    if (!function) {
        static void *handle = NULL;
        if (!handle) {
            handle = dlopen(NULL, RTLD_LAZY);
        }
        function = handle ? (GLFMProc)dlsym(handle, functionName) : NULL;
    }
    return function;
```

```
}

void glfmSetKeyboardVisible(GLFMDisplay *display, bool visible) {
    GLFMPlatformData *platformData = (GLFMPlatformData *)display->platformData;
    if (_glfmSetKeyboardVisible(platformData, visible)) {
        if (visible && display->uiChrome == GLFMUserInterfaceChromeFullscreen) {
            // This seems to be required to reset to fullscreen when the keyboard is shown.
            _glfmSetFullScreen(platformData->app, GLFMUserInterfaceChromeNavigationAndStatusBar);
        }
    }
}

bool glfmIsKeyboardVisible(GLFMDisplay *display) {
    GLFMPlatformData *platformData = (GLFMPlatformData *)display->platformData;
    return platformData->keyboardVisible;
}

// MARK: Android-specific public functions

ANativeActivity *glfmAndroidGetActivity() {
    if (platformDataGlobal && platformDataGlobal->app) {
        return platformDataGlobal->app->activity;
    } else {
        return NULL;
    }
}

#endif
```

44 src/glfm_platform_emscripten.c

```
/*
 GLFM
 https://github.com/brackeen/glfm
 Copyright (c) 2014-2017 David Brackeen

 This software is provided 'as-is', without any express or implied warranty.
 In no event will the authors be held liable for any damages arising from the
 use of this software. Permission is granted to anyone to use this software
 for any purpose, including commercial applications, and to alter it and
 redistribute it freely, subject to the following restrictions:

 1. The origin of this software must not be misrepresented; you must not
    claim that you wrote the original software. If you use this software in a
    product, an acknowledgment in the product documentation would be appreciated
    but is not required.
 2. Altered source versions must be plainly marked as such, and must not be
    misrepresented as being the original software.
 3. This notice may not be removed or altered from any source distribution.
 */

#include "glfm.h"

#ifdef GLFM_PLATFORM_EMSCRIPTEN
```

```
25  #include <EGL/egl.h>
    #include <emscripten/emscripten.h>
    #include <emscripten/html5.h>
    #include <math.h>
    #include <stdlib.h>
30  #include <sys/time.h>
    #include <time.h>

    #include "glfm_platform.h"

35  #define MAX_ACTIVE_TOUCHES 10

    typedef struct {
        long identifier;
        bool active;
40  } GLFMActiveTouch;

    typedef struct {
        bool multitouchEnabled;
        int32_t width;
45      int32_t height;
        double scale;
        GLFMRenderingAPI renderingAPI;

        bool mouseDown;
50      GLFMActiveTouch activeTouches[MAX_ACTIVE_TOUCHES];

        bool active;
        bool isFullscreen;
    } GLFMPlatformData;
55
    static void _glfmClearActiveTouches(GLFMPlatformData *platformData);

    // MARK: GLFM implementation

60  void glfmSetUserInterfaceOrientation(GLFMDisplay *display,
                                         GLFMUserInterfaceOrientation ↙
                                             ↳ allowedOrientations) {
        if (display->allowedOrientations != allowedOrientations) {
            display->allowedOrientations = allowedOrientations;

65          // Lock orientation
            // NOTE: I'm not sure this works anywhere yet
            if (allowedOrientations == GLFMUserInterfaceOrientationPortrait) {
                emscripten_lock_orientation(EMSCRIPTEN_ORIENTATION_PORTRAIT_PRIMARY ↙
                    ↳ |
                                            EMSCRIPTEN_ORIENTATION_PORTRAIT_SECONDARY↙
                                                ↳ );
70          } else if (allowedOrientations == GLFMUserInterfaceOrientationLandscape)↙
                ↳ {
                emscripten_lock_orientation(EMSCRIPTEN_ORIENTATION_LANDSCAPE_PRIMARY↙
                    ↳ |
                                            EMSCRIPTEN_ORIENTATION_LANDSCAPE_SECONDARY↙
                                                ↳ );
            } else {
                emscripten_lock_orientation(EMSCRIPTEN_ORIENTATION_PORTRAIT_PRIMARY ↙
                    ↳ |
```

```
                                                    EMSCRIPTEN_ORIENTATION_PORTRAIT_SECONDARY↙
                                                        ↳ |
                                                    EMSCRIPTEN_ORIENTATION_LANDSCAPE_PRIMARY↙
                                                        ↳ |
                                                    EMSCRIPTEN_ORIENTATION_LANDSCAPE_SECONDARY↙
                                                        ↳ );
            }
        }
    }

    void glfmGetDisplaySize(GLFMDisplay *display, int *width, int *height) {
        GLFMPlatformData *platformData = display->platformData;
        *width = platformData->width;
        *height = platformData->height;
    }

    double glfmGetDisplayScale(GLFMDisplay *display) {
        GLFMPlatformData *platformData = display->platformData;
        return platformData->scale;
    }

    void glfmGetDisplayChromeInsets(GLFMDisplay *display, double *top, double *right↙
        ↳ , double *bottom,
                                    double *left) {
        GLFMPlatformData *platformData = display->platformData;

        *top = platformData->scale * EM_ASM_DOUBLE_V( {
            var htmlStyles = window.getComputedStyle(document.querySelector("html"))↙
                ↳ ;
            return (parseInt(htmlStyles.getPropertyValue("--glfm-chrome-top-old")) ↙
                ↳ || 0) +
                (parseInt(htmlStyles.getPropertyValue("--glfm-chrome-top")) || 0)↙
                    ↳ ;
        } );
        *right = platformData->scale * EM_ASM_DOUBLE_V( {
            var htmlStyles = window.getComputedStyle(document.querySelector("html"))↙
                ↳ ;
            return (parseInt(htmlStyles.getPropertyValue("--glfm-chrome-right-old"))↙
                ↳ || 0) +
                (parseInt(htmlStyles.getPropertyValue("--glfm-chrome-right")) || ↙
                    ↳ 0);
        } );
        *bottom = platformData->scale * EM_ASM_DOUBLE_V( {
            var htmlStyles = window.getComputedStyle(document.querySelector("html"))↙
                ↳ ;
            return (parseInt(htmlStyles.getPropertyValue("--glfm-chrome-bottom-old")↙
                ↳ ) || 0) +
                (parseInt(htmlStyles.getPropertyValue("--glfm-chrome-bottom")) ||↙
                    ↳ 0);
        } );
        *left = platformData->scale * EM_ASM_DOUBLE_V( {
            var htmlStyles = window.getComputedStyle(document.querySelector("html"))↙
                ↳ ;
            return (parseInt(htmlStyles.getPropertyValue("--glfm-chrome-left-old")) ↙
                ↳ || 0) +
                (parseInt(htmlStyles.getPropertyValue("--glfm-chrome-left")) || ↙
                    ↳ 0);
```

```
    } );
}

void _glfmDisplayChromeUpdated(GLFMDisplay *display) {
    GLFMPlatformData *platformData = display->platformData;

    if (display->uiChrome == GLFMUserInterfaceChromeFullscreen) {
        if (!platformData->isFullscreen) {
            EMSCRIPTEN_RESULT result = emscripten_request_fullscreen(NULL, ↙
                ↳ EM_FALSE);
            platformData->isFullscreen = (result == EMSCRIPTEN_RESULT_SUCCESS);
            if (!platformData->isFullscreen) {
                display->uiChrome = GLFMUserInterfaceChromeNavigation;
            }
        }
    } else if (platformData->isFullscreen) {
        platformData->isFullscreen = false;
        emscripten_exit_fullscreen();
    }
}

GLFMRenderingAPI glfmGetRenderingAPI(GLFMDisplay *display) {
    GLFMPlatformData *platformData = display->platformData;
    return platformData->renderingAPI;
}

bool glfmHasTouch(GLFMDisplay *display) {
    (void)display;
    return EM_ASM_INT_V({
        return (('ontouchstart' in window) || (navigator.msMaxTouchPoints > 0));
    });
}

void glfmSetMouseCursor(GLFMDisplay *display, GLFMMouseCursor mouseCursor) {
    (void)display;
    // Make sure the javascript array emCursors is refernced properly
    int emCursor = 0;
    switch (mouseCursor) {
        case GLFMMouseCursorAuto:
            emCursor = 0;
            break;
        case GLFMMouseCursorNone:
            emCursor = 1;
            break;
        case GLFMMouseCursorDefault:
            emCursor = 2;
            break;
        case GLFMMouseCursorPointer:
            emCursor = 3;
            break;
        case GLFMMouseCursorCrosshair:
            emCursor = 4;
            break;
        case GLFMMouseCursorText:
            emCursor = 5;
            break;
    }
```

```
    EM_ASM_({
        var emCursors = new Array('auto', 'none', 'default', 'pointer', '↙
            ↳ crosshair', 'text');
        Module['canvas'].style.cursor = emCursors[$0];
    },
        emCursor);
}

void glfmSetMultitouchEnabled(GLFMDisplay *display, bool multitouchEnabled) {
    GLFMPlatformData *platformData = display->platformData;
    platformData->multitouchEnabled = multitouchEnabled;
}

bool glfmGetMultitouchEnabled(GLFMDisplay *display) {
    GLFMPlatformData *platformData = display->platformData;
    return platformData->multitouchEnabled;
}

void glfmSetKeyboardVisible(GLFMDisplay *display, bool visible) {
    (void)display;
    (void)visible;
    // Do nothing
}

bool glfmIsKeyboardVisible(GLFMDisplay *display) {
    (void)display;
    return false;
}

GLFMProc glfmGetProcAddress(const char *functionName) {
    return eglGetProcAddress(functionName);
}

// MARK: Emscripten glue

static int _glfmGetDisplayWidth(GLFMDisplay *display) {
    (void)display;
    const double width = EM_ASM_DOUBLE_V({
        var canvas = Module['canvas'];
        return canvas.width;
    });
    return (int)(round(width));
}

static int _glfmGetDisplayHeight(GLFMDisplay *display) {
    (void)display;
    const double height = EM_ASM_DOUBLE_V({
        var canvas = Module['canvas'];
        return canvas.height;
    });
    return (int)(round(height));
}

static void _glfmSetActive(GLFMDisplay *display, bool active) {
    GLFMPlatformData *platformData = display->platformData;
    if (platformData->active != active) {
        platformData->active = active;
```

```
            _glfmClearActiveTouches(platformData);
            if (display->focusFunc) {
                display->focusFunc(display, active);
            }
        }
    }

    static void _glfmMainLoopFunc(void *userData) {
        GLFMDisplay *display = userData;
        if (display) {
            // Check if canvas size has changed
            int displayChanged = EM_ASM_INT_V({
                var canvas = Module['canvas'];
                var devicePixelRatio = window.devicePixelRatio || 1;
                var width = canvas.clientWidth * devicePixelRatio;
                var height = canvas.clientHeight * devicePixelRatio;
                if (width != canvas.width || height != canvas.height) {
                    canvas.width = width;
                    canvas.height = height;
                    return 1;
                } else {
                    return 0;
                }
            });
            if (displayChanged) {
                GLFMPlatformData *platformData = display->platformData;
                platformData->width = _glfmGetDisplayWidth(display);
                platformData->height = _glfmGetDisplayHeight(display);
                platformData->scale = emscripten_get_device_pixel_ratio();
                if (display->surfaceResizedFunc) {
                    display->surfaceResizedFunc(display, platformData->width, ↙
                        ↳ platformData->height);
                }
            }

            // Tick
            if (display->mainLoopFunc) {
                // NOTE: The JavaScript requestAnimationFrame callback sends the ↙
                    ↳ frame time as a
                // parameter, but Emscripten include send it.
                display->mainLoopFunc(display, emscripten_get_now() / 1000.0);
            }
        }
    }

    static EM_BOOL _glfmWebGLContextCallback(int eventType, const void *reserved, ↙
        ↳ void *userData) {
        (void)reserved;
        GLFMDisplay *display = userData;
        if (eventType == EMSCRIPTEN_EVENT_WEBGLCONTEXTLOST) {
            if (display->surfaceDestroyedFunc) {
                display->surfaceDestroyedFunc(display);
            }
            return 1;
        } else if (eventType == EMSCRIPTEN_EVENT_WEBGLCONTEXTRESTORED) {
            GLFMPlatformData *platformData = display->platformData;
            if (display->surfaceCreatedFunc) {
```

```
            display->surfaceCreatedFunc(display, platformData->width,
                ↳ platformData->height);
        }
        return 1;
    } else {
        return 0;
    }
}

static EM_BOOL _glfmVisibilityChangeCallback(int eventType,
                                             const
                                                 ↳ EmscriptenVisibilityChangeEvent
                                                 ↳ *e,
                                             void *userData) {
    (void)eventType;
    GLFMDisplay *display = userData;
    _glfmSetActive(display, !e->hidden);
    return 1;
}

static EM_BOOL _glfmKeyCallback(int eventType, const EmscriptenKeyboardEvent *e,
    ↳ void *userData) {
    GLFMDisplay *display = userData;
    if (eventType == EMSCRIPTEN_EVENT_KEYPRESS) {
        if (display->charFunc && e->charCode >= ' ') {
            display->charFunc(display, e->key, 0);
        }
        return 1;
    }
    // Prevent change of focus via tab key
    EM_BOOL handled = e->keyCode == '\t';
    if (display->keyFunc) {
        GLFMKeyAction action;
        if (eventType == EMSCRIPTEN_EVENT_KEYDOWN) {
            if (e->repeat) {
                action = GLFMKeyActionRepeated;
            } else {
                action = GLFMKeyActionPressed;
            }
        } else if (eventType == EMSCRIPTEN_EVENT_KEYUP) {
            action = GLFMKeyActionReleased;
        } else {
            return 0;
        }

        /*
         TODO: Modifiers
         For now just send e->keyCode as is. It is identical to the defined
             ↳ values:
         GLFMKeyBackspace = 0x08,
         GLFMKeyTab       = 0x09,
         GLFMKeyEnter     = 0x0d,
         GLFMKeyEscape    = 0x1b,
         GLFMKeySpace     = 0x20,
         GLFMKeyLeft      = 0x25,
         GLFMKeyUp        = 0x26,
         GLFMKeyRight     = 0x27,
```

```
            GLFMKeyDown       = 0x28,
335         */

            return display->keyFunc(display, e->keyCode, action, 0) || handled;
        } else {
            return handled;
340     }
    }

    static EM_BOOL _glfmMouseCallback(int eventType, const EmscriptenMouseEvent *e, ↙
        ↳ void *userData) {
        GLFMDisplay *display = userData;
345     if (display->touchFunc) {
            GLFMPlatformData *platformData = display->platformData;
            GLFMTouchPhase touchPhase;
            switch (eventType) {
                case EMSCRIPTEN_EVENT_MOUSEDOWN:
350                 touchPhase = GLFMTouchPhaseBegan;
                    platformData->mouseDown = true;
                    break;

                case EMSCRIPTEN_EVENT_MOUSEMOVE:
355                 if (platformData->mouseDown) {
                        touchPhase = GLFMTouchPhaseMoved;
                    } else {
                        touchPhase = GLFMTouchPhaseHover;
                    }
360                 break;

                case EMSCRIPTEN_EVENT_MOUSEUP:
                    touchPhase = GLFMTouchPhaseEnded;
                    platformData->mouseDown = false;
365                 break;

                default:
                    touchPhase = GLFMTouchPhaseCancelled;
                    platformData->mouseDown = false;
370                 break;
            }
            return display->touchFunc(display, e->button, touchPhase,
                                      platformData->scale * (double)e->canvasX,
                                      platformData->scale * (double)e->canvasY);
375     } else {
            return 0;
        }
    }

380 static void _glfmClearActiveTouches(GLFMPlatformData *platformData) {
        for (int i = 0; i < MAX_ACTIVE_TOUCHES; i++) {
            platformData->activeTouches[i].active = false;
        }
    }
385
    static int _glfmGetTouchIdentifier(GLFMPlatformData *platformData, const ↙
        ↳ EmscriptenTouchPoint *t) {
        int firstNullIndex = -1;
        int index = -1;
```

```
    for (int i = 0; i < MAX_ACTIVE_TOUCHES; i++) {
        if (platformData->activeTouches[i].identifier == t->identifier &&
            platformData->activeTouches[i].active) {
            index = i;
            break;
        } else if (firstNullIndex == -1 && !platformData->activeTouches[i].
            ↪ active) {
            firstNullIndex = i;
        }
    }
    if (index == -1) {
        if (firstNullIndex == -1) {
            // Shouldn't happen
            return -1;
        }
        index = firstNullIndex;
        platformData->activeTouches[index].identifier = t->identifier;
        platformData->activeTouches[index].active = true;
    }
    return index;
}

static EM_BOOL _glfmTouchCallback(int eventType, const EmscriptenTouchEvent *e,
    ↪ void *userData) {
    GLFMDisplay *display = userData;
    if (display->touchFunc) {
        GLFMPlatformData *platformData = display->platformData;
        GLFMTouchPhase touchPhase;
        switch (eventType) {
            case EMSCRIPTEN_EVENT_TOUCHSTART:
                touchPhase = GLFMTouchPhaseBegan;
                break;

            case EMSCRIPTEN_EVENT_TOUCHMOVE:
                touchPhase = GLFMTouchPhaseMoved;
                break;

            case EMSCRIPTEN_EVENT_TOUCHEND:
                touchPhase = GLFMTouchPhaseEnded;
                break;

            case EMSCRIPTEN_EVENT_TOUCHCANCEL:
            default:
                touchPhase = GLFMTouchPhaseCancelled;
                break;
        }

        int handled = 0;
        for (int i = 0; i < e->numTouches; i++) {
            const EmscriptenTouchPoint *t = &e->touches[i];
            if (t->isChanged) {
                int identifier = _glfmGetTouchIdentifier(platformData, t);
                if (identifier >= 0) {
                    if ((platformData->multitouchEnabled || identifier == 0)) {
                        handled |= display->touchFunc(display, identifier,
                            ↪ touchPhase,
                                                      platformData->scale * (
```

```
                                                                    ↪ double)t->canvasX,
                                                                platformData->scale * (↙
                                                                    ↪ double)t->canvasY);
                        }

                        if (touchPhase == GLFMTouchPhaseEnded || touchPhase == ↙
                            ↪ GLFMTouchPhaseCancelled) {
                            platformData->activeTouches[identifier].active = false;
                        }
                    }
                }
            }
            return handled;
        } else {
            return 0;
        }
    }

    // MARK: main

    int main() {
        GLFMDisplay *glfmDisplay = calloc(1, sizeof(GLFMDisplay));
        GLFMPlatformData *platformData = calloc(1, sizeof(GLFMPlatformData));
        glfmDisplay->platformData = platformData;
        platformData->active = true;
        _glfmClearActiveTouches(platformData);

        // Main entry
        glfmMain(glfmDisplay);

        // Init resizable canvas
        EM_ASM({
            var canvas = Module['canvas'];
            var devicePixelRatio = window.devicePixelRatio || 1;
            canvas.width = canvas.clientWidth * devicePixelRatio;
            canvas.height = canvas.clientHeight * devicePixelRatio;
        });
        platformData->width = _glfmGetDisplayWidth(glfmDisplay);
        platformData->height = _glfmGetDisplayHeight(glfmDisplay);
        platformData->scale = emscripten_get_device_pixel_ratio();

        // Create WebGL context
        EmscriptenWebGLContextAttributes attribs;
        emscripten_webgl_init_context_attributes(&attribs);
        attribs.alpha = glfmDisplay->colorFormat == GLFMColorFormatRGBA8888;
        attribs.depth = glfmDisplay->depthFormat != GLFMDepthFormatNone;
        attribs.stencil = glfmDisplay->stencilFormat != GLFMStencilFormatNone;
        attribs.antialias = glfmDisplay->multisample != GLFMMultisampleNone;
        attribs.premultipliedAlpha = 1;
        attribs.preserveDrawingBuffer = 0;
        attribs.preferLowPowerToHighPerformance = 0;
        attribs.failIfMajorPerformanceCaveat = 0;
        attribs.enableExtensionsByDefault = 0;

        int contextHandle = 0;
        if (glfmDisplay->preferredAPI >= GLFMRenderingAPIOpenGLES3) {
            // OpenGL ES 3.0 / WebGL 2.0
```

```
            attribs.majorVersion = 2;
            attribs.minorVersion = 0;
            contextHandle = emscripten_webgl_create_context(NULL, &attribs);
            if (contextHandle) {
                platformData->renderingAPI = GLFMRenderingAPIOpenGLES3;
            }
        }
        if (!contextHandle) {
            // OpenGL ES 2.0 / WebGL 1.0
            attribs.majorVersion = 1;
            attribs.minorVersion = 0;
            contextHandle = emscripten_webgl_create_context(NULL, &attribs);
            if (contextHandle) {
                platformData->renderingAPI = GLFMRenderingAPIOpenGLES2;
            }
        }
        if (!contextHandle) {
            _glfmReportSurfaceError(glfmDisplay, "Couldn't create GL context");
            return 0;
        }

        emscripten_webgl_make_context_current(contextHandle);

        if (glfmDisplay->surfaceCreatedFunc) {
            glfmDisplay->surfaceCreatedFunc(glfmDisplay, platformData->width, ↙
                ↳ platformData->height);
        }

        // Setup callbacks
        emscripten_set_main_loop_arg(_glfmMainLoopFunc, glfmDisplay, 0, 0);
        emscripten_set_touchstart_callback(0, glfmDisplay, 1, _glfmTouchCallback);
        emscripten_set_touchend_callback(0, glfmDisplay, 1, _glfmTouchCallback);
        emscripten_set_touchmove_callback(0, glfmDisplay, 1, _glfmTouchCallback);
        emscripten_set_touchcancel_callback(0, glfmDisplay, 1, _glfmTouchCallback);
        emscripten_set_mousedown_callback(0, glfmDisplay, 1, _glfmMouseCallback);
        emscripten_set_mouseup_callback(0, glfmDisplay, 1, _glfmMouseCallback);
        emscripten_set_mousemove_callback(0, glfmDisplay, 1, _glfmMouseCallback);
        //emscripten_set_click_callback(0, 0, 1, mouse_callback);
        //emscripten_set_dblclick_callback(0, 0, 1, mouse_callback);
        emscripten_set_keypress_callback(0, glfmDisplay, 1, _glfmKeyCallback);
        emscripten_set_keydown_callback(0, glfmDisplay, 1, _glfmKeyCallback);
        emscripten_set_keyup_callback(0, glfmDisplay, 1, _glfmKeyCallback);
        emscripten_set_webglcontextlost_callback(0, glfmDisplay, 1, ↙
            ↳ _glfmWebGLContextCallback);
        emscripten_set_webglcontextrestored_callback(0, glfmDisplay, 1, ↙
            ↳ _glfmWebGLContextCallback);
        emscripten_set_visibilitychange_callback(glfmDisplay, 1, ↙
            ↳ _glfmVisibilityChangeCallback);
        return 0;
    }

    #endif
```

45 src/glfm_platform.h

```
/*
 GLFM
```

```
    https://github.com/brackeen/glfm
    Copyright (c) 2014-2017 David Brackeen

    This software is provided 'as-is', without any express or implied warranty.
    In no event will the authors be held liable for any damages arising from the
    use of this software. Permission is granted to anyone to use this software
    for any purpose, including commercial applications, and to alter it and
    redistribute it freely, subject to the following restrictions:

    1. The origin of this software must not be misrepresented; you must not
       claim that you wrote the original software. If you use this software in a
       product, an acknowledgment in the product documentation would be appreciated
       but is not required.
    2. Altered source versions must be plainly marked as such, and must not be
       misrepresented as being the original software.
    3. This notice may not be removed or altered from any source distribution.
    */

#ifndef GLFM_PLATFORM_H
#define GLFM_PLATFORM_H

#include "glfm.h"
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef __cplusplus
extern "C" {
#endif

struct GLFMDisplay {
    // Config
    GLFMRenderingAPI preferredAPI;
    GLFMColorFormat colorFormat;
    GLFMDepthFormat depthFormat;
    GLFMStencilFormat stencilFormat;
    GLFMMultisample multisample;
    GLFMUserInterfaceOrientation allowedOrientations;
    GLFMUserInterfaceChrome uiChrome;

    // Callbacks
    GLFMMainLoopFunc mainLoopFunc;
    GLFMTouchFunc touchFunc;
    GLFMKeyFunc keyFunc;
    GLFMCharFunc charFunc;
    GLFMSurfaceErrorFunc surfaceErrorFunc;
    GLFMSurfaceCreatedFunc surfaceCreatedFunc;
    GLFMSurfaceResizedFunc surfaceResizedFunc;
    GLFMSurfaceDestroyedFunc surfaceDestroyedFunc;
    GLFMKeyboardVisibilityChangedFunc keyboardVisibilityChangedFunc;
    GLFMMemoryWarningFunc lowMemoryFunc;
    GLFMAppFocusFunc focusFunc;

    // External data
    void *userData;
    void *platformData;
```

```
};

// MARK: Setters

void glfmSetSurfaceErrorFunc(GLFMDisplay *display, GLFMSurfaceErrorFunc surfaceErrorFunc) {
    if (display) {
        display->surfaceErrorFunc = surfaceErrorFunc;
    }
}

void glfmSetDisplayConfig(GLFMDisplay *display,
                          const GLFMRenderingAPI preferredAPI,
                          const GLFMColorFormat colorFormat,
                          const GLFMDepthFormat depthFormat,
                          const GLFMStencilFormat stencilFormat,
                          const GLFMMultisample multisample) {
    if (display) {
        display->preferredAPI = preferredAPI;
        display->colorFormat = colorFormat;
        display->depthFormat = depthFormat;
        display->stencilFormat = stencilFormat;
        display->multisample = multisample;
    }
}

GLFMUserInterfaceOrientation glfmGetUserInterfaceOrientation(GLFMDisplay *display) {
    return display ? display->allowedOrientations : GLFMUserInterfaceOrientationAny;
}

void glfmSetUserData(GLFMDisplay *display, void *userData) {
    if (display) {
        display->userData = userData;
    }
}

void *glfmGetUserData(GLFMDisplay *display) {
    return display ? display->userData : NULL;
}

void _glfmDisplayChromeUpdated(GLFMDisplay *display);

GLFMUserInterfaceChrome glfmGetDisplayChrome(GLFMDisplay *display) {
    return display ? display->uiChrome : GLFMUserInterfaceChromeNavigation;
}

void glfmSetDisplayChrome(GLFMDisplay *display, GLFMUserInterfaceChrome uiChrome) {
    if (display) {
        display->uiChrome = uiChrome;
        _glfmDisplayChromeUpdated(display);
    }
}

void glfmSetMainLoopFunc(GLFMDisplay *display, GLFMMainLoopFunc mainLoopFunc) {
```

```
    if (display) {
        display->mainLoopFunc = mainLoopFunc;
    }
}

void glfmSetSurfaceCreatedFunc(GLFMDisplay *display, GLFMSurfaceCreatedFunc surfaceCreatedFunc) {
    if (display) {
        display->surfaceCreatedFunc = surfaceCreatedFunc;
    }
}

void glfmSetSurfaceResizedFunc(GLFMDisplay *display, GLFMSurfaceResizedFunc surfaceResizedFunc) {
    if (display) {
        display->surfaceResizedFunc = surfaceResizedFunc;
    }
}

void glfmSetSurfaceDestroyedFunc(GLFMDisplay *display,
                                 GLFMSurfaceDestroyedFunc surfaceDestroyedFunc) {
    if (display) {
        display->surfaceDestroyedFunc = surfaceDestroyedFunc;
    }
}

void glfmSetKeyboardVisibilityChangedFunc(GLFMDisplay *display,
                                          GLFMKeyboardVisibilityChangedFunc func) {
    if (display) {
        display->keyboardVisibilityChangedFunc = func;
    }
}

void glfmSetTouchFunc(GLFMDisplay *display, GLFMTouchFunc touchFunc) {
    if (display) {
        display->touchFunc = touchFunc;
    }
}

void glfmSetKeyFunc(GLFMDisplay *display, GLFMKeyFunc keyFunc) {
    if (display) {
        display->keyFunc = keyFunc;
    }
}

void glfmSetCharFunc(GLFMDisplay *display, GLFMCharFunc charFunc) {
    if (display) {
        display->charFunc = charFunc;
    }
}

void glfmSetMemoryWarningFunc(GLFMDisplay *display, GLFMMemoryWarningFunc lowMemoryFunc) {
    if (display) {
        display->lowMemoryFunc = lowMemoryFunc;
```

```
    }
}

void glfmSetAppFocusFunc(GLFMDisplay *display, GLFMAppFocusFunc focusFunc) {
    if (display) {
        display->focusFunc = focusFunc;
    }
}

// MARK: Helper functions

static void _glfmReportSurfaceError(GLFMDisplay *display, const char *↙
    ↳ errorMessage) {
    if (display->surfaceErrorFunc && errorMessage) {
        display->surfaceErrorFunc(display, errorMessage);
    }
}

// glfmExtensionSupported function is from
// http://www.opengl.org/archives/resources/features/OGLextensions/
bool glfmExtensionSupported(const char *extension) {
    // Extension names should not have spaces.
    GLubyte *where = (GLubyte *)strchr(extension, ' ');
    if (where || *extension == '\0') {
        return false;
    }

    const GLubyte *extensions = glGetString(GL_EXTENSIONS);

    // It takes a bit of care to be fool-proof about parsing the
    // OpenGL extensions string. Don't be fooled by sub-strings, etc.
    const GLubyte *start = extensions;
    for (;;) {
        where = (GLubyte *)strstr((const char *)start, extension);
        if (!where) {
            break;
        }

        GLubyte *terminator = where + strlen(extension);
        if (where == start || *(where - 1) == ' ') {
            if (*terminator == ' ' || *terminator == '\0') {
                return true;
            }
        }

        start = terminator;
    }

    return false;
}

#ifdef __cplusplus
}
#endif

#endif
```

46 src/glfm_platform_ios.m

```
/*
 GLFM
 https://github.com/brackeen/glfm
 Copyright (c) 2014-2017 David Brackeen

 This software is provided 'as-is', without any express or implied warranty.
 In no event will the authors be held liable for any damages arising from the
 use of this software. Permission is granted to anyone to use this software
 for any purpose, including commercial applications, and to alter it and
 redistribute it freely, subject to the following restrictions:

 1. The origin of this software must not be misrepresented; you must not
    claim that you wrote the original software. If you use this software in a
    product, an acknowledgment in the product documentation would be appreciated
    but is not required.
 2. Altered source versions must be plainly marked as such, and must not be
    misrepresented as being the original software.
 3. This notice may not be removed or altered from any source distribution.
 */

#include "glfm.h"

#if defined(GLFM_PLATFORM_IOS) || defined(GLFM_PLATFORM_TVOS)

#import <UIKit/UIKit.h>

#include <dlfcn.h>
#include "glfm_platform.h"

#define MAX_SIMULTANEOUS_TOUCHES 10

#define CHECK_GL_ERROR() do { GLenum error = glGetError(); if (error != ↙
    ↳ GL_NO_ERROR) \
NSLog(@"OpenGL error 0x%04x at glfm_platform_ios.m:%i", error, __LINE__); } ↙
    ↳ while(0)

@interface GLFMAppDelegate : NSObject <UIApplicationDelegate>

@property(nonatomic, strong) UIWindow *window;
@property(nonatomic, assign) BOOL active;

@end

#pragma mark - GLFMView

@interface GLFMView : UIView {
    GLint _drawableWidth;
    GLint _drawableHeight;
    GLuint _defaultFramebuffer;
    GLuint _colorRenderbuffer;
    GLuint _attachmentRenderbuffer;
    GLuint _msaaFramebuffer;
    GLuint _msaaRenderbuffer;
}
```

```
@property(nonatomic, strong) EAGLContext *context;
@property(nonatomic, assign) NSString *colorFormat;
@property(nonatomic, assign) BOOL preserveBackbuffer;
@property(nonatomic, assign) NSUInteger depthBits;
@property(nonatomic, assign) NSUInteger stencilBits;
@property(nonatomic, assign) BOOL multisampling;
@property(nonatomic, readonly) NSUInteger drawableWidth;
@property(nonatomic, readonly) NSUInteger drawableHeight;

- (void)createDrawable;
- (void)deleteDrawable;
- (void)prepareRender;
- (void)finishRender;

@end

@implementation GLFMView

@dynamic drawableWidth, drawableHeight;

+ (Class)layerClass {
    return [CAEAGLLayer class];
}

- (void)dealloc {
    [self deleteDrawable];
}

- (NSUInteger)drawableWidth {
    return (NSUInteger)_drawableWidth;
}

- (NSUInteger)drawableHeight {
    return (NSUInteger)_drawableHeight;
}

- (void)createDrawable {
    if (_defaultFramebuffer != 0 || !self.context) {
        return;
    }

    if (!self.colorFormat) {
        self.colorFormat = kEAGLColorFormatRGBA8;
    }

    [EAGLContext setCurrentContext:self.context];

    CAEAGLLayer *eaglLayer = (CAEAGLLayer *)self.layer;
    eaglLayer.opaque = YES;
    eaglLayer.drawableProperties =
        @{ kEAGLDrawablePropertyRetainedBacking : @(self.preserveBackbuffer),
           kEAGLDrawablePropertyColorFormat : self.colorFormat };

    glGenFramebuffers(1, &_defaultFramebuffer);
    glGenRenderbuffers(1, &_colorRenderbuffer);
    glBindFramebuffer(GL_FRAMEBUFFER, _defaultFramebuffer);
    glBindRenderbuffer(GL_RENDERBUFFER, _colorRenderbuffer);
```

```
    // iPhone 6 Display Zoom hack - use a modified bounds so that the renderbufferStorage method
    // creates the correct size renderbuffer.
    CGRect oldBounds = eaglLayer.bounds;
    if (eaglLayer.contentsScale == 2.343750) {
        if (eaglLayer.bounds.size.width == 320.0 && eaglLayer.bounds.size.height == 568.0) {
            eaglLayer.bounds = CGRectMake(eaglLayer.bounds.origin.x, eaglLayer.bounds.origin.y,
                                          eaglLayer.bounds.size.width,
                                          1334 / eaglLayer.contentsScale);
        } else if (eaglLayer.bounds.size.width == 568.0 && eaglLayer.bounds.size.height == 320.0) {
            eaglLayer.bounds = CGRectMake(eaglLayer.bounds.origin.x, eaglLayer.bounds.origin.y,
                                          1334 / eaglLayer.contentsScale,
                                          eaglLayer.bounds.size.height);
        }
    }

    if (![self.context renderbufferStorage:GL_RENDERBUFFER fromDrawable:eaglLayer]) {
        NSLog(@"Error: Call to renderbufferStorage failed");
    }

    eaglLayer.bounds = oldBounds;

    glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_RENDERBUFFER,
                              _colorRenderbuffer);

    glGetRenderbufferParameteriv(GL_RENDERBUFFER, GL_RENDERBUFFER_WIDTH, &_drawableWidth);
    glGetRenderbufferParameteriv(GL_RENDERBUFFER, GL_RENDERBUFFER_HEIGHT, &_drawableHeight);

    if (_multisampling) {
        glGenFramebuffers(1, &_msaaFramebuffer);
        glBindFramebuffer(GL_FRAMEBUFFER, _msaaFramebuffer);

        glGenRenderbuffers(1, &_msaaRenderbuffer);
        glBindRenderbuffer(GL_RENDERBUFFER, _msaaRenderbuffer);

        GLenum internalformat = GL_RGBA8_OES;
        if ([kEAGLColorFormatRGB565 isEqualToString:_colorFormat]) {
            internalformat = GL_RGB565;
        }

        glRenderbufferStorageMultisampleAPPLE(GL_RENDERBUFFER, 4, internalformat,
                                              _drawableWidth, _drawableHeight);

        glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_RENDERBUFFER,
                                  _msaaRenderbuffer);
```

```
            GLenum status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
            if (status != GL_FRAMEBUFFER_COMPLETE) {
                NSLog(@"Error: Couldn't create multisample framebuffer: 0x%04x", ↙
                    ↳ status);
            }
        }

        if (_depthBits > 0 || _stencilBits > 0) {
            glGenRenderbuffers(1, &_attachmentRenderbuffer);
            glBindRenderbuffer(GL_RENDERBUFFER, _attachmentRenderbuffer);

            GLenum internalformat;
            if (_depthBits > 0 && _stencilBits > 0) {
                internalformat = GL_DEPTH24_STENCIL8_OES;
            } else if (_depthBits >= 24) {
                internalformat = GL_DEPTH_COMPONENT24_OES;
            } else if (_depthBits > 0) {
                internalformat = GL_DEPTH_COMPONENT16;
            } else {
                internalformat = GL_STENCIL_INDEX8;
            }

            if (_multisampling) {
                glRenderbufferStorageMultisampleAPPLE(GL_RENDERBUFFER, 4, ↙
                    ↳ internalformat,
                                                      _drawableWidth, ↙
                                                          ↳ _drawableHeight);
            } else {
                glRenderbufferStorage(GL_RENDERBUFFER, internalformat, ↙
                    ↳ _drawableWidth, _drawableHeight);
            }

            if (_depthBits > 0) {
                glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, ↙
                    ↳ GL_RENDERBUFFER,
                                          _attachmentRenderbuffer);
            }
            if (_stencilBits > 0) {
                glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_STENCIL_ATTACHMENT, ↙
                    ↳ GL_RENDERBUFFER,
                                          _attachmentRenderbuffer);
            }
        }

        GLenum status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
        if (status != GL_FRAMEBUFFER_COMPLETE) {
            NSLog(@"Error: Framebuffer incomplete: 0x%04x", status);
        }

        CHECK_GL_ERROR();
    }

    - (void)deleteDrawable {
        if (_defaultFramebuffer) {
            glDeleteFramebuffers(1, &_defaultFramebuffer);
            _defaultFramebuffer = 0;
        }
```

```
    if (_colorRenderbuffer) {
        glDeleteRenderbuffers(1, &_colorRenderbuffer);
        _colorRenderbuffer = 0;
    }
    if (_attachmentRenderbuffer) {
        glDeleteRenderbuffers(1, &_attachmentRenderbuffer);
        _attachmentRenderbuffer = 0;
    }
    if (_msaaRenderbuffer) {
        glDeleteRenderbuffers(1, &_msaaRenderbuffer);
        _msaaRenderbuffer = 0;
    }
    if (_msaaFramebuffer) {
        glDeleteFramebuffers(1, &_msaaFramebuffer);
        _msaaFramebuffer = 0;
    }
}

- (void)prepareRender {
    [EAGLContext setCurrentContext:self.context];
    if (_multisampling) {
        glBindFramebuffer(GL_FRAMEBUFFER, _msaaFramebuffer);
        glBindRenderbuffer(GL_RENDERBUFFER, _msaaRenderbuffer);
    } else {
        glBindFramebuffer(GL_FRAMEBUFFER, _defaultFramebuffer);
        glBindRenderbuffer(GL_RENDERBUFFER, _colorRenderbuffer);
    }
    CHECK_GL_ERROR();
}

- (void)finishRender {
    if (_multisampling) {
        glBindFramebuffer(GL_READ_FRAMEBUFFER_APPLE, _msaaFramebuffer);
        glBindFramebuffer(GL_DRAW_FRAMEBUFFER_APPLE, _defaultFramebuffer);
        glResolveMultisampleFramebufferAPPLE();
    }

    static bool checked_GL_EXT_discard_framebuffer = false;
    static bool has_GL_EXT_discard_framebuffer = false;
    if (!checked_GL_EXT_discard_framebuffer) {
        checked_GL_EXT_discard_framebuffer = true;
        if (glfmExtensionSupported("GL_EXT_discard_framebuffer")) {
            has_GL_EXT_discard_framebuffer = true;
        }
    }
    if (has_GL_EXT_discard_framebuffer) {
        GLenum target = GL_FRAMEBUFFER;
        GLenum attachments[3];
        GLsizei numAttachments = 0;
        if (_multisampling) {
            target = GL_READ_FRAMEBUFFER_APPLE;
            attachments[numAttachments++] = GL_COLOR_ATTACHMENT0;
        }
        if (_depthBits > 0) {
            attachments[numAttachments++] = GL_DEPTH_ATTACHMENT;
        }
        if (_stencilBits > 0) {
```

```
265                 attachments[numAttachments++] = GL_STENCIL_ATTACHMENT;
            }
            if (numAttachments > 0) {
                if (_multisampling) {
                    glBindFramebuffer(GL_FRAMEBUFFER, _msaaFramebuffer);
270                 } else {
                    glBindFramebuffer(GL_FRAMEBUFFER, _defaultFramebuffer);
                }
                glDiscardFramebufferEXT(target, numAttachments, attachments);
            }
275     }

        glBindRenderbuffer(GL_RENDERBUFFER, _colorRenderbuffer);
        [self.context presentRenderbuffer:GL_RENDERBUFFER];

280     CHECK_GL_ERROR();
    }

    - (void)layoutSubviews {
        CGSize size = self.preferredDrawableSize;
285     NSUInteger newDrawableWidth = (NSUInteger)size.width;
        NSUInteger newDrawableHeight = (NSUInteger)size.height;

        if (self.drawableWidth != newDrawableWidth || self.drawableHeight != ↙
            ↳ newDrawableHeight) {
            [self deleteDrawable];
290         [self createDrawable];
        }
    }

    - (CGSize)preferredDrawableSize {
295     NSUInteger newDrawableWidth = (NSUInteger)(self.bounds.size.width * self.↙
            ↳ contentScaleFactor);
        NSUInteger newDrawableHeight = (NSUInteger)(self.bounds.size.height * self.↙
            ↳ contentScaleFactor);

        // On the iPhone 6 when "Display Zoom" is set, the size will be incorrect.
        if (self.contentScaleFactor == 2.343750) {
300         if (newDrawableWidth == 750 && newDrawableHeight == 1331) {
                newDrawableHeight = 1334;
            } else if (newDrawableWidth == 1331 && newDrawableHeight == 750) {
                newDrawableWidth = 1334;
            }
305     }

        return CGSizeMake(newDrawableWidth, newDrawableHeight);
    }

310 @end

    #pragma mark - GLFMViewController

    @interface GLFMViewController : UIViewController<UIKeyInput, UITextInputTraits> ↙
        ↳ {
315     const void *activeTouches[MAX_SIMULTANEOUS_TOUCHES];
    }
```

```
@property(nonatomic, strong) EAGLContext *context;
@property(nonatomic, strong) CADisplayLink *displayLink;
@property(nonatomic, assign) GLFMDisplay *glfmDisplay;
@property(nonatomic, assign) CGSize drawableSize;
@property(nonatomic, assign) BOOL multipleTouchEnabled;
@property(nonatomic, assign) BOOL keyboardRequested;
@property(nonatomic, assign) BOOL keyboardVisible;
@property(nonatomic, assign) BOOL surfaceCreatedNotified;

@end

@implementation GLFMViewController

- (id)init {
    if ((self = [super init])) {
        [self clearTouches];
        _glfmDisplay = calloc(1, sizeof(GLFMDisplay));
        _glfmDisplay->platformData = (__bridge void *)self;
    }
    return self;
}

- (BOOL)animating {
    return (self.displayLink != nil);
}

- (void)setAnimating:(BOOL)animating {
    if (self.animating != animating) {
        if (!animating) {
            [self.displayLink invalidate];
            self.displayLink = nil;
        } else {
            self.displayLink = [CADisplayLink displayLinkWithTarget:self
                                                            selector:@selector(render:)];
            [self.displayLink addToRunLoop:[NSRunLoop mainRunLoop] forMode:NSRunLoopCommonModes];
        }
    }
}

- (BOOL)prefersStatusBarHidden {
    return _glfmDisplay->uiChrome != GLFMUserInterfaceChromeNavigationAndStatusBar;
}

- (BOOL)prefersHomeIndicatorAutoHidden {
    return _glfmDisplay->uiChrome == GLFMUserInterfaceChromeFullscreen;
}

- (void)loadView {
    GLFMAppDelegate *delegate = UIApplication.sharedApplication.delegate;
    self.view = [[GLFMView alloc] initWithFrame:delegate.window.bounds];
    self.view.autoresizingMask = UIViewAutoresizingFlexibleWidth | UIViewAutoresizingFlexibleHeight;
    self.view.contentScaleFactor = [UIScreen mainScreen].nativeScale;
}
```

```
- (void)viewDidLoad {
    [super viewDidLoad];

    GLFMView *view = (GLFMView *)self.view;
    self.drawableSize = [view preferredDrawableSize];

    glfmMain(_glfmDisplay);

    if (_glfmDisplay->preferredAPI >= GLFMRenderingAPIOpenGLES3) {
        self.context = [[EAGLContext alloc] initWithAPI:↙
            ↳ kEAGLRenderingAPIOpenGLES3];
    }
    if (!self.context) {
        self.context = [[EAGLContext alloc] initWithAPI:↙
            ↳ kEAGLRenderingAPIOpenGLES2];
    }

    if (!self.context) {
        _glfmReportSurfaceError(_glfmDisplay, "Failed to create ES context");
        return;
    }

    view.context = self.context;

#if TARGET_OS_IOS
    view.multipleTouchEnabled = self.multipleTouchEnabled;

    [self setNeedsStatusBarAppearanceUpdate];
#endif

    switch (_glfmDisplay->colorFormat) {
        case GLFMColorFormatRGB565:
            view.colorFormat = kEAGLColorFormatRGB565;
            break;
        case GLFMColorFormatRGBA8888:
        default:
            view.colorFormat = kEAGLColorFormatRGBA8;
            break;
    }

    switch (_glfmDisplay->depthFormat) {
        case GLFMDepthFormatNone:
        default:
            view.depthBits = 0;
            break;
        case GLFMDepthFormat16:
            view.depthBits = 16;
            break;
        case GLFMDepthFormat24:
            view.depthBits = 24;
            break;
    }

    switch (_glfmDisplay->stencilFormat) {
        case GLFMStencilFormatNone:
        default:
```

```
                view.stencilBits = 0;
                break;
            case GLFMStencilFormat8:
                view.stencilBits = 8;
                break;
        }

        view.multisampling = _glfmDisplay->multisample != GLFMMultisampleNone;

        [view createDrawable];

        if (view.drawableWidth > 0 && view.drawableHeight > 0) {
            self.drawableSize = CGSizeMake(view.drawableWidth, view.drawableHeight);
            self.animating = YES;
        }

    #if TARGET_OS_IOS
        [NSNotificationCenter.defaultCenter addObserver:self selector:@selector(
            ↪ keyboardFrameChanged:)
                                                   name:
                                                       ↪ UIKeyboardWillChangeFrameNotification
                                                       ↪
                                                 object:view.window];
    #endif
    }

    - (UIInterfaceOrientationMask)supportedInterfaceOrientations {
        BOOL isTablet = (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad);
        GLFMUserInterfaceOrientation uiOrientations = _glfmDisplay->
            ↪ allowedOrientations;
        if (uiOrientations == GLFMUserInterfaceOrientationAny) {
            if (isTablet) {
                return UIInterfaceOrientationMaskAll;
            } else {
                return UIInterfaceOrientationMaskAllButUpsideDown;
            }
        } else if (uiOrientations == GLFMUserInterfaceOrientationPortrait) {
            if (isTablet) {
                return (UIInterfaceOrientationMask)(
                    ↪ UIInterfaceOrientationMaskPortrait |
                        UIInterfaceOrientationMaskPortraitUpsideDown);
            } else {
                return UIInterfaceOrientationMaskPortrait;
            }
        } else {
            return UIInterfaceOrientationMaskLandscape;
        }
    }

    - (void)didReceiveMemoryWarning {
        [super didReceiveMemoryWarning];
        if (_glfmDisplay->lowMemoryFunc) {
            _glfmDisplay->lowMemoryFunc(_glfmDisplay);
        }
    }

    - (void)dealloc {
```

```
    [self setAnimating:NO];
    if ([EAGLContext currentContext] == self.context) {
        [EAGLContext setCurrentContext:nil];
    }
    if (_glfmDisplay->surfaceDestroyedFunc) {
        _glfmDisplay->surfaceDestroyedFunc(_glfmDisplay);
    }
    free(_glfmDisplay);
}

- (void)render:(CADisplayLink *)displayLink {
    GLFMView *view = (GLFMView *)self.view;

    if (!self.surfaceCreatedNotified) {
        self.surfaceCreatedNotified = YES;

        if (_glfmDisplay->surfaceCreatedFunc) {
            _glfmDisplay->surfaceCreatedFunc(_glfmDisplay, (int)self.↙
                ↳ drawableSize.width,
                                             (int)self.drawableSize.height);
        }
    }

    CGSize newDrawableSize = CGSizeMake(view.drawableWidth, view.drawableHeight)↙
        ↳ ;
    if (!CGSizeEqualToSize(self.drawableSize, newDrawableSize)) {
        self.drawableSize = newDrawableSize;
        if (_glfmDisplay->surfaceResizedFunc) {
            _glfmDisplay->surfaceResizedFunc(_glfmDisplay, (int)self.↙
                ↳ drawableSize.width,
                                             (int)self.drawableSize.height);
        }
    }

    [view prepareRender];
    if (_glfmDisplay->mainLoopFunc) {
        _glfmDisplay->mainLoopFunc(_glfmDisplay, displayLink.timestamp);
    }
    [view finishRender];
}

#pragma mark - UIResponder

- (void)clearTouches {
    for (int i = 0; i < MAX_SIMULTANEOUS_TOUCHES; i++) {
        activeTouches[i] = NULL;
    }
}

- (void)addTouchEvent:(UITouch *)touch withType:(GLFMTouchPhase)phase {
    int firstNullIndex = -1;
    int index = -1;
    for (int i = 0; i < MAX_SIMULTANEOUS_TOUCHES; i++) {
        if (activeTouches[i] == (__bridge const void *)touch) {
            index = i;
            break;
        } else if (firstNullIndex == -1 && activeTouches[i] == NULL) {
```

```
                firstNullIndex = i;
            }
        }
        if (index == -1) {
            if (firstNullIndex == -1) {
                // Shouldn't happen
                return;
            }
            index = firstNullIndex;
            activeTouches[index] = (__bridge const void *)touch;
        }

        if (_glfmDisplay->touchFunc) {
            CGPoint currLocation = [touch locationInView:self.view];
            currLocation.x *= self.view.contentScaleFactor;
            currLocation.y *= self.view.contentScaleFactor;

            _glfmDisplay->touchFunc(_glfmDisplay, index, phase,
                                    (double)currLocation.x, (double)currLocation.y);
        }

        if (phase == GLFMTouchPhaseEnded || phase == GLFMTouchPhaseCancelled) {
            activeTouches[index] = NULL;
        }
    }

    - (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
        for (UITouch *touch in touches) {
            [self addTouchEvent:touch withType:GLFMTouchPhaseBegan];
        }
    }

    - (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
        for (UITouch *touch in touches) {
            [self addTouchEvent:touch withType:GLFMTouchPhaseMoved];
        }
    }

    - (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
        for (UITouch *touch in touches) {
            [self addTouchEvent:touch withType:GLFMTouchPhaseEnded];
        }
    }

    - (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event {
        for (UITouch *touch in touches) {
            [self addTouchEvent:touch withType:GLFMTouchPhaseCancelled];
        }
    }

    #if TARGET_OS_TV

    - (BOOL)handlePress:(UIPress *)press withAction:(GLFMKeyAction)action {
        if (_glfmDisplay->keyFunc) {
            GLFMKey key = (GLFMKey)0;
            switch (press.type) {
                case UIPressTypeUpArrow:
```

```
                key = GLFMKeyUp;
                break;
            case UIPressTypeDownArrow:
                key = GLFMKeyDown;
                break;
            case UIPressTypeLeftArrow:
                key = GLFMKeyLeft;
                break;
            case UIPressTypeRightArrow:
                key = GLFMKeyRight;
                break;
            case UIPressTypeSelect:
                key = GLFMKeyNavSelect;
                break;
            case UIPressTypeMenu:
                key = GLFMKeyNavMenu;
                break;
            case UIPressTypePlayPause:
                key = GLFMKeyPlayPause;
                break;
        }
        if (key != 0) {
            return _glfmDisplay->keyFunc(_glfmDisplay, key, action, 0);
        } else {
            return NO;
        }
    } else {
        return NO;
    }
}

- (void)pressesBegan:(NSSet<UIPress *> *)presses withEvent:(UIPressesEvent *)event {
    BOOL handled = YES;
    for (UIPress *press in presses) {
        handled &= [self handlePress:press withAction:GLFMKeyActionPressed];
    }
    if (!handled) {
        [super pressesBegan:presses withEvent:event];
    }
}

- (void)pressesChanged:(NSSet<UIPress *> *)presses withEvent:(UIPressesEvent *)event {
    [super pressesChanged:presses withEvent:event];
}

- (void)pressesEnded:(NSSet<UIPress *> *)presses withEvent:(UIPressesEvent *)event {
    BOOL handled = YES;
    for (UIPress *press in presses) {
        handled &= [self handlePress:press withAction:GLFMKeyActionReleased];
    }
    if (!handled) {
        [super pressesEnded:presses withEvent:event];
    }
}
```

```
- (void)pressesCancelled:(NSSet<UIPress *> *)presses withEvent:(UIPressesEvent *)event {
    [super pressesCancelled:presses withEvent:event];
}

#endif

#pragma mark - UIKeyInput

#if TARGET_OS_IOS

- (void)keyboardFrameChanged:(NSNotification *)notification {
    id value = notification.userInfo[UIKeyboardFrameEndUserInfoKey];
    if ([value isKindOfClass:[NSValue class]]) {
        NSValue *nsValue = value;
        CGRect keyboardFrame = [nsValue CGRectValue];

        self.keyboardVisible = CGRectIntersectsRect(self.view.window.frame, keyboardFrame);

        if (_glfmDisplay->keyboardVisibilityChangedFunc) {
            // Convert to view coordinates
            keyboardFrame = [self.view convertRect:keyboardFrame fromView:nil];

            // Convert to pixels
            keyboardFrame.origin.x *= self.view.contentScaleFactor;
            keyboardFrame.origin.y *= self.view.contentScaleFactor;
            keyboardFrame.size.width *= self.view.contentScaleFactor;
            keyboardFrame.size.height *= self.view.contentScaleFactor;

            _glfmDisplay->keyboardVisibilityChangedFunc(_glfmDisplay, self.keyboardVisible,
                                                        keyboardFrame.origin.x,
                                                        keyboardFrame.origin.y,
                                                        keyboardFrame.size.width,
                                                        keyboardFrame.size.height);
        }
    }
}

#endif

// UITextInputTraits - disable suggestion bar
- (UITextAutocorrectionType)autocorrectionType {
    return UITextAutocorrectionTypeNo;
}

- (BOOL)hasText {
    return YES;
}

- (void)insertText:(NSString *)text {
    if (_glfmDisplay->charFunc) {
        _glfmDisplay->charFunc(_glfmDisplay, text.UTF8String, 0);
```

```
695         }
    }

    - (void)deleteBackward {
        if (_glfmDisplay->keyFunc) {
700             _glfmDisplay->keyFunc(_glfmDisplay, GLFMKeyBackspace, ↙
                    ↳ GLFMKeyActionPressed, 0);
                _glfmDisplay->keyFunc(_glfmDisplay, GLFMKeyBackspace, ↙
                    ↳ GLFMKeyActionReleased, 0);
        }
    }

705 - (BOOL)canBecomeFirstResponder {
        return self.keyboardRequested;
    }

    - (NSArray *)keyCommands {
710     static NSArray *keyCommands = NULL;
        if (!keyCommands) {
            keyCommands = @[ [UIKeyCommand keyCommandWithInput:UIKeyInputUpArrow
                                                 modifierFlags:(UIKeyModifierFlags)0
                                                        action:@selector(keyPressed↙
                                                            ↳ :)],
715                          [UIKeyCommand keyCommandWithInput:UIKeyInputDownArrow
                                                 modifierFlags:(UIKeyModifierFlags)0
                                                        action:@selector(keyPressed↙
                                                            ↳ :)],
                             [UIKeyCommand keyCommandWithInput:UIKeyInputLeftArrow
                                                 modifierFlags:(UIKeyModifierFlags)0
720                                                    action:@selector(keyPressed↙
                                                            ↳ :)],
                             [UIKeyCommand keyCommandWithInput:UIKeyInputRightArrow
                                                 modifierFlags:(UIKeyModifierFlags)0
                                                        action:@selector(keyPressed↙
                                                            ↳ :)],
                             [UIKeyCommand keyCommandWithInput:UIKeyInputEscape
725                                          modifierFlags:(UIKeyModifierFlags)0
                                                        action:@selector(keyPressed↙
                                                            ↳ :)] ];
        };

        return keyCommands;
730 }

    - (void)keyPressed:(UIKeyCommand *)keyCommand {
        if (_glfmDisplay->keyFunc) {
            NSString *key = [keyCommand input];
735         GLFMKey keyCode = (GLFMKey)0;
            if (key == UIKeyInputUpArrow) {
                keyCode = GLFMKeyUp;
            } else if (key == UIKeyInputDownArrow) {
                keyCode = GLFMKeyDown;
740         } else if (key == UIKeyInputLeftArrow) {
                keyCode = GLFMKeyLeft;
            } else if (key == UIKeyInputRightArrow) {
                keyCode = GLFMKeyRight;
            } else if (key == UIKeyInputEscape) {
```

```
745                 keyCode = GLFMKeyEscape;
            }

            if (keyCode != 0) {
                _glfmDisplay->keyFunc(_glfmDisplay, keyCode, GLFMKeyActionPressed, ↙
                    ↳ 0);
750                 _glfmDisplay->keyFunc(_glfmDisplay, keyCode, GLFMKeyActionReleased, ↙
                    ↳ 0);
            }
        }
    }

755 @end

    #pragma mark - Application Delegate

    @implementation GLFMAppDelegate
760
    - (BOOL)application:(UIApplication *)application
        didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
        _active = YES;
        self.window = [[UIWindow alloc] init];
765     if (self.window.bounds.size.width <= 0.0 || self.window.bounds.size.height ↙
            ↳ <= 0.0) {
            // Set UIWindow frame for iOS 8.
            // On iOS 9, the UIWindow frame may be different than the UIScreen ↙
                ↳ bounds for iPad's
            // Split View or Slide Over.
            self.window.frame = [[UIScreen mainScreen] bounds];
770     }
        self.window.rootViewController = [[GLFMViewController alloc] init];
        [self.window makeKeyAndVisible];
        return YES;
    }
775
    - (void)setActive:(BOOL)active {
        if (_active != active) {
            _active = active;

780         GLFMViewController *vc = (GLFMViewController *)[self.window ↙
                ↳ rootViewController];
            [vc clearTouches];
            vc.animating = active;
            if (vc.glfmDisplay && vc.glfmDisplay->focusFunc) {
                vc.glfmDisplay->focusFunc(vc.glfmDisplay, _active);
785         }
        }
    }

    - (void)applicationWillResignActive:(UIApplication *)application {
790     self.active = NO;
    }

    - (void)applicationDidEnterBackground:(UIApplication *)application {
        self.active = NO;
795 }
```

```
- (void)applicationWillEnterForeground:(UIApplication *)application {
    self.active = YES;
}

- (void)applicationDidBecomeActive:(UIApplication *)application {
    self.active = YES;
}

- (void)applicationWillTerminate:(UIApplication *)application {
    self.active = NO;
}

@end

#pragma mark - Main

int main(int argc, char *argv[]) {
    @autoreleasepool {
        return UIApplicationMain(0, NULL, nil, NSStringFromClass([GLFMAppDelegate class]));
    }
}

#pragma mark - GLFM implementation

GLFMProc glfmGetProcAddress(const char *functionName) {
    static void *handle = NULL;
    if (!handle) {
        handle = dlopen(NULL, RTLD_LAZY);
    }
    return handle ? (GLFMProc)dlsym(handle, functionName) : NULL;
}

void glfmSetUserInterfaceOrientation(GLFMDisplay *display,
                                     GLFMUserInterfaceOrientation allowedOrientations) {
    if (display) {
        if (display->allowedOrientations != allowedOrientations) {
            display->allowedOrientations = allowedOrientations;

            // HACK: Notify that the value of supportedInterfaceOrientations has changed
            GLFMViewController *vc = (__bridge GLFMViewController *)display->platformData;
            if (vc.view.window) {
                UIViewController *dummyVC = [[UIViewController alloc] init];
                dummyVC.view = [[UIView alloc] init];
                [vc presentViewController:dummyVC animated:NO completion:^{
                    [vc dismissViewControllerAnimated:NO completion:NULL];
                }];
            }
        }
    }
}

void glfmGetDisplaySize(GLFMDisplay *display, int *width, int *height) {
    if (display && display->platformData) {
```

```
        GLFMViewController *vc = (__bridge GLFMViewController *)display->↙
            ↳ platformData;
        *width = (int)vc.drawableSize.width;
        *height = (int)vc.drawableSize.height;
    } else {
        *width = 0;
        *height = 0;
    }
}

double glfmGetDisplayScale(GLFMDisplay *display) {
    if (display && display->platformData) {
        GLFMViewController *vc = (__bridge GLFMViewController *)display->↙
            ↳ platformData;
        return vc.view.contentScaleFactor;
    } else {
        return [UIScreen mainScreen].scale;
    }
}

void glfmGetDisplayChromeInsets(GLFMDisplay *display, double *top, double *right↙
    ↳ , double *bottom,
                                double *left) {
    if (display && display->platformData) {
        GLFMViewController *vc = (__bridge GLFMViewController *)display->↙
            ↳ platformData;
        if (@available(iOS 11, tvOS 11, *)) {
            UIEdgeInsets insets = vc.view.safeAreaInsets;
            *top = insets.top * vc.view.contentScaleFactor;
            *right = insets.right * vc.view.contentScaleFactor;
            *bottom = insets.bottom * vc.view.contentScaleFactor;
            *left = insets.left * vc.view.contentScaleFactor;
        } else {
#if TARGET_OS_IOS
            if (![vc prefersStatusBarHidden]) {
                *top = ([UIApplication sharedApplication].statusBarFrame.size.↙
                    ↳ height *
                        vc.view.contentScaleFactor);
            } else {
                *top = 0.0;
            }
#else
            *top = 0.0;
#endif
            *right = 0.0;
            *bottom = 0.0;
            *left = 0.0;
        }
    } else {
        *top = 0.0;
        *right = 0.0;
        *bottom = 0.0;
        *left = 0.0;
    }
}

void _glfmDisplayChromeUpdated(GLFMDisplay *display) {
```

```
    if (display && display->platformData) {
#if TARGET_OS_IOS
        GLFMViewController *vc = (__bridge GLFMViewController *)display->platformData;
        [vc setNeedsStatusBarAppearanceUpdate];
        if (@available(iOS 11, *)) {
            [vc setNeedsUpdateOfHomeIndicatorAutoHidden];
        }
#endif
    }
}

GLFMRenderingAPI glfmGetRenderingAPI(GLFMDisplay *display) {
    if (display && display->platformData) {
        GLFMViewController *vc = (__bridge GLFMViewController *)display->platformData;
        if (vc.context.API == kEAGLRenderingAPIOpenGLES3) {
            return GLFMRenderingAPIOpenGLES3;
        } else {
            return GLFMRenderingAPIOpenGLES2;
        }
    } else {
        return GLFMRenderingAPIOpenGLES2;
    }
}

bool glfmHasTouch(GLFMDisplay *display) {
    return true;
}

void glfmSetMouseCursor(GLFMDisplay *display, GLFMMouseCursor mouseCursor) {
    // Do nothing
}

void glfmSetMultitouchEnabled(GLFMDisplay *display, bool multitouchEnabled) {
    if (display) {
#if TARGET_OS_IOS
        GLFMViewController *vc = (__bridge GLFMViewController *)display->platformData;
        vc.multipleTouchEnabled = (BOOL)multitouchEnabled;
        if (vc.isViewLoaded) {
            vc.view.multipleTouchEnabled = (BOOL)multitouchEnabled;
        }
#endif
    }
}

bool glfmGetMultitouchEnabled(GLFMDisplay *display) {
    if (display) {
        GLFMViewController *vc = (__bridge GLFMViewController *)display->platformData;
        return vc.multipleTouchEnabled;
    } else {
        return false;
    }
}
```

```
955 void glfmSetKeyboardVisible(GLFMDisplay *display, bool visible) {
    if (display) {
        GLFMViewController *vc = (__bridge GLFMViewController *)display->↙
            ↳ platformData;
        vc.keyboardRequested = visible;
        if (visible) {
960         [vc becomeFirstResponder];
        } else {
            [vc resignFirstResponder];
        }
    }
965 }

    bool glfmIsKeyboardVisible(GLFMDisplay *display) {
        if (display) {
            GLFMViewController *vc = (__bridge GLFMViewController *)display->↙
                ↳ platformData;
970         return vc.keyboardRequested;
        } else {
            return false;
        }
    }
975
    #endif
```