Skip to content

Navigation Menu

google

filament

- Code
- Issues165

Filament is a real-time physically based rendering engine for Android, iOS, Windows, Linux, macOS, and WebGL2

google.github.io/filament/

License

Apache-2.0 license

Code of conduct

Code of conduct

Contributing

Contributing

Security policy

Security policy

18.8k stars 2k forks 385 watching 144 Branches 240 Tags Activity Custom properties

Public repository

google/filament

Name	
pixelflinger	
2 days ago	
.github	last week
android	3 days ago
art	2 months ago
assets	2 years ago
build	last week
docs	2 days ago

docs_src	2 days ago
filament	2 days ago
ide	8 months ago
ios	2 weeks ago

Repository files navigation

- README
- Code of conduct

Filament

Filament is a real-time physically based rendering engine for Android, iOS, Linux, macOS, Windows, and WebGL. It is designed to be as small as possible and as efficient as possible on Android.

Download

Download Filament releases to access stable builds. Filament release archives contains host-side tools that are required to generate assets.

Make sure you always use tools from the same release as the runtime library. This is particularly important for `matc` (material compiler).

If you'd rather build Filament yourself, please refer to our build manual

Android

Android projects can simply declare Filament libraries as Maven dependencies:

```
repositories {
    // ...
    mavenCentral()
}

dependencies {
    implementation 'com.google.android.filament:filament-android:1.64.0'
}
```

Here are all the libraries available in the group `com.google.android.filament`:

Artifact	Description
	The Filament rendering engine itself.
	Debug version of `filament-android`.
	A glTF 2.0 loader for Filament, depends on `filament-android`.
	KTX loading, Kotlin math, and camera utilities, depends on `gltfio-android`.

	A runtime material builder/compiler. This library is large but contains a full shader compiler/validator/optimizer and supports both OpenGL and Vulkan.
	A much smaller alternative to `filamat-android` that can only generate OpenGL shaders. It does not provide validation or optimizations.

iOS

iOS projects can use CocoaPods to install the latest release:

```
pod 'Filament', '~> 1.64.0'
```

Documentation

- Filament: an in-depth explanation of real-time physically based rendering, the graphics capabilities and implementation of Filament. This document explains the math and reasoning behind most of our decisions. This document is a good introduction to PBR for graphics programmers.
- Materials: the full reference documentation for our material system. This document explains our different material models, how to use the material compiler `matc` and how to write custom materials.
- Material Properties: a reference sheet for the standard material model.

Examples

Features

APIs

- Native C++ API for Android, iOS, Linux, macOS and Windows
- Java/JNI API for Android
- JavaScript API

Backends

- OpenGL 4.1+ for Linux, macOS and Windows
- OpenGL ES 3.0+ for Android and iOS
- Metal for macOS and iOS
- Vulkan 1.0 for Android, Linux, macOS, and Windows
- WebGL 2.0 for all platforms

Rendering

- Clustered forward renderer
- Cook-Torrance microfacet specular BRDF
- Lambertian diffuse BRDF
- Custom lighting/surface shading
- HDR/linear lighting
- Metallic workflow
- Clear coat
- Anisotropic lighting
- Approximated translucent (subsurface) materials
- Cloth/fabric/sheen shading
- Normal mapping & ambient occlusion mapping
- Image-based lighting
- Physically-based camera (shutter speed, sensitivity and aperture)
- Physical light units
- Point lights, spot lights, and directional light

- Specular anti-aliasing
- Point, spot, and directional light shadows
- Cascaded shadows
- EVSM, PCSS, DPCF, or PCF shadows
- Transparent shadows
- Contact shadows
- Screen-space ambient occlusion
- Screen-space reflections
- Screen-space refraction
- Global fog
- Dynamic resolution (with support for AMD FidelityFX FSR)

Post processing

- HDR bloom
- Depth of field bokeh
- Multiple tone mappers: generic (customizable), ACES, filmic, etc.
- Color and tone management: luminance scaling, gamut mapping
- Color grading: exposure, night adaptation, white balance, channel mixer, shadows/mid-tones/highlights, ASC CDL, contrast, saturation, etc.
- TAA, FXAA, MSAA
- Screen-space lens flares

glTF 2.0

- Encodings
    - Embeded
    - Binary
- Primitive Types
    - Points
    - Lines
    - Line Loop
    - Line Strip
    - Triangles
    - Triangle Strip
    - Triangle Fan
- Animation
    - Transform animation
    - Linear interpolation
    - Morph animation

        - Sparse accessor
        - Skin animation
        - Joint animation
    - Extensions
        - KHR_draco_mesh_compression
        - KHR_lights_punctual
        - KHR_materials_clearcoat
        - KHR_materials_emissive_strength
        - KHR_materials_ior
        - KHR_materials_pbrSpecularGlossiness
        - KHR_materials_sheen
        - KHR_materials_transmission
        - KHR_materials_unlit
        - KHR_materials_variants
        - KHR_materials_volume
        - KHR_materials_specular
        - KHR_mesh_quantization
        - KHR_texture_basisu
        - KHR_texture_transform
        - EXT_meshopt_compression

Rendering with Filament

Native Linux, macOS and Windows

You must create an `Engine`, a `Renderer` and a `SwapChain`. The `SwapChain` is created from a native window pointer (an `NSView` on macOS or a `HWND` on Windows for instance):

```
Engine* engine = Engine::create();
SwapChain* swapChain = engine->createSwapChain(nativeWindow);
Renderer* renderer = engine->createRenderer();
```

To render a frame you must then create a `View`, a `Scene` and a `Camera`:

```
Camera* camera = engine->createCamera(EntityManager::get().create());
View* view = engine->createView();
```

```
    Scene* scene = engine->createScene();

    view->setCamera(camera);
view->setScene(scene);
```

Renderables are added to the scene:

```
    Entity renderable = EntityManager::get().create();
    // build a quad
    RenderableManager::Builder(1)
            .boundingBox({{ -1, -1, -1 }, { 1, 1, 1 }})
            .material(0, materialInstance)
            .geometry(0, RenderableManager::PrimitiveType::TRIANGLES,
    vertexBuffer, indexBuffer, 0, 6)
            .culling(false)
            .build(*engine, renderable);
scene->addEntity(renderable);
```

The material instance is obtained from a material, itself loaded from a binary blob generated by `matc`:

```
    Material* material = Material::Builder()
            .package((void*) BAKED_MATERIAL_PACKAGE,
    sizeof(BAKED_MATERIAL_PACKAGE))
            .build(*engine);
MaterialInstance* materialInstance = material->createInstance();
```

To learn more about materials and `matc`, please refer to the materials documentation

To render, simply pass the `View` to the `Renderer`:

```
    // beginFrame() returns false if we need to skip a frame
    if (renderer->beginFrame(swapChain)) {
        // for each View
        renderer->render(view);
        renderer->endFrame();
}
```

For complete examples of Linux, macOS and Windows Filament applications, look at the source files in the `samples/` directory. These samples are all based on `libs/filamentapp/` which contains the code that creates a native window with SDL2 and initializes the Filament engine, renderer and views.

For more information on how to prepare environment maps for image-based lighting please refer to BUILDING.md

Android

See `android/samples` for examples of how to use Filament on Android.

You must always first initialize Filament by calling `Filament.init()`.

Rendering with Filament on Android is similar to rendering from native code (the APIs are largely the same across languages). You can render into a `Surface` by passing a `Surface` to the `createSwapChain` method. This allows you to render to a `SurfaceTexture`, a `TextureView` or a `SurfaceView`. To make things easier we provide an Android specific API called `UiHelper` in the package `com.google.android.filament.android`. All you need to do is set a render callback on the helper and attach your `SurfaceView` or `TextureView` to it. You are still responsible for creating the swap chain in the `onNativeWindowChanged()` callback.

iOS

Filament is supported on iOS 11.0 and above. See `ios/samples` for examples of using Filament on iOS.

Filament on iOS is largely the same as native rendering with C++. A `CAEAGLLayer` or `CAMetalLayer` is passed to the `createSwapChain` method. Filament for iOS supports both Metal (preferred) and OpenGL ES.

Assets

To get started you can use the textures and environment maps found respectively in `third_party/textures` and `third_party/environments`. These assets are under CC0 license. Please refer to their respective `URL.txt` files to know more about the original authors.

Environments must be pre-processed using `cmgen` or using the `libiblprefilter` library.

How to make contributions

Please read and follow the steps in CONTRIBUTING.md. Make sure you are familiar with the code style

Directory structure

This repository not only contains the core Filament engine, but also its supporting libraries and tools.

- `android`: Android libraries and projects
  - `filamat-android`: Filament material generation library (AAR) for Android
  - `filament-android`: Filament library (AAR) for Android
  - `filament-utils-android`: Extra utilities (KTX loader, math types, etc.)
  - `gltfio-android`: Filament glTF loading library (AAR) for Android
  - `samples`: Android-specific Filament samples
- `art`: Source for various artworks (logos, PDF manuals, etc.)
- `assets`: 3D assets to use with sample applications
- `build`: CMake build scripts
- `docs`: Documentation
  - `math`: Mathematica notebooks used to explore BRDFs, equations, etc.
- `filament`: Filament rendering engine (minimal dependencies)

    - `backend`: Rendering backends/drivers (Vulkan, Metal, OpenGL/ES)
- `ide`: Configuration files for IDEs (CLion, etc.)
- `ios`: Sample projects for iOS
- `libs`: Libraries
    - `bluegl`: OpenGL bindings for macOS, Linux and Windows
    - `bluevk`: Vulkan bindings for macOS, Linux, Windows and Android
    - `camutils`: Camera manipulation utilities
    - `filabridge`: Library shared by the Filament engine and host tools
    - `filaflat`: Serialization/deserialization library used for materials
    - `filagui`: Helper library for Dear ImGui
    - `filamat`: Material generation library
    - `filamentapp`: SDL2 skeleton to build sample apps
    - `filameshio`: Tiny filamesh parsing library (see also `tools/filamesh`)
    - `geometry`: Mesh-related utilities
    - `gltfio`: Loader for glTF 2.0
    - `ibl`: IBL generation tools
    - `image`: Image filtering and simple transforms
    - `imageio`: Image file reading / writing, only intended for internal use
    - `matdbg`: DebugServer for inspecting shaders at run-time (debug builds only)
    - `math`: Math library
    - `mathio`: Math types support for output streams
    - `utils`: Utility library (threads, memory, data structures, etc.)
    - `viewer`: glTF viewer library (requires gltfio)
- `samples`: Sample desktop applications
- `shaders`: Shaders used by `filamat` and `matc`
- `third_party`: External libraries and assets
    - `environments`: Environment maps under CC0 license that can be used with `cmgen`
    - `models`: Models under permissive licenses
    - `textures`: Textures under CC0 license
- `tools`: Host tools
    - `cmgen`: Image-based lighting asset generator
    - `filamesh`: Mesh converter
    - `glslminifier`: Minifies GLSL source code
    - `matc`: Material compiler
    - `filament-matp`: Material parser
    - `matinfo` Displays information about materials compiled with `matc`
    - `mipgen` Generates a series of miplevels from a source image
    - `normal-blending`: Tool to blend normal maps
    - `resgen` Aggregates binary blobs into embeddable resources
    - `roughness-prefilter`: Pre-filters a roughness map from a normal map to reduce aliasing

- `specular-color`: Computes the specular color of conductors based on spectral data
- `web`: JavaScript bindings, documentation, and samples

License

Please see LICENSE.

Disclaimer

This is not an officially supported Google product.

Releases 235

v1.64.0 Latest

last week

+ 234 releases

Contributors 195

+ 181 contributors

Languages

- C++58.1%
- Assembly14.6%
- C10.3%
- Java4.4%
- HTML2.3%
- Kotlin2.3%
- Other8.0%

Footer

© 2025 GitHub, Inc.

Footer navigation

- Terms
- Privacy
- Security
- Status
- Docs
- Contact
- Manage cookies
- Do not share my personal information