How rendering works in Ebitengine

hajimehoshi@gmail.com
Last Update: 2024-05-07
Status: Public

TL;DR

`Image` in the package `ebiten` (github.com/hajimehoshi/ebiten) is the core of Ebitengine's API for rendering. This API is simple, but graphics operations via the API are not directly translated into commands for a low-level graphics layer. One of the reasons is efficiency, and besides the performance, there are a lot of requirements for rendering. To solve the issues, Ebitengine adopted a layered rendering system.

Background

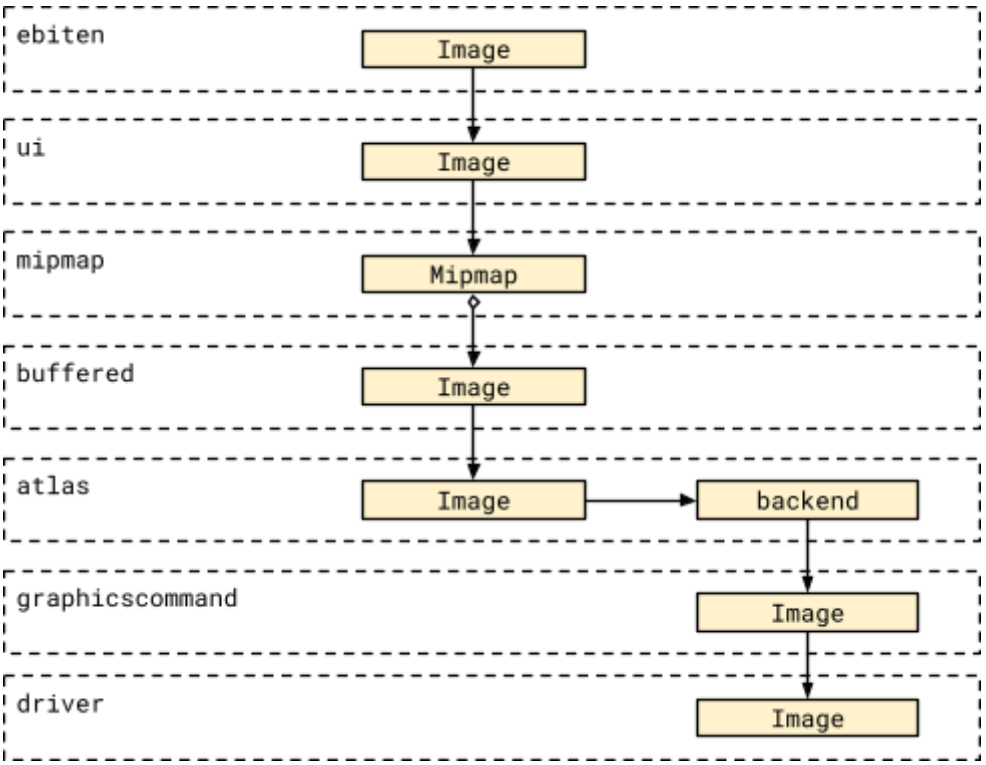
Ebitengine offers a very simple rendering API with `Image`. On the other hand, actual rendering is not so simple. Ebitengine needs to satisfy these requirements:

- For efficient rendering, the number of graphics commands that are sent to GPU should be small. However, we don't want Ebitengine users to care about low-layer things. Ebitengine should efficiently translate graphics operations to a small number of GPU graphics commands automatically.
- There are some environments that cannot start rendering before the main loop due to gomobile restrictions. Then, graphics operations before the main loop needs to be buffered somewhere.
- When the linear filter is used, mipmap images are preferable.

To achieve this, Ebitengine adopted a layered rendering system.

Overview

This figure explains the layered system of Ebitengine rendering at the latest commit.



ebiten.Image

An Image in the package ebiten is a user-facing API for rendering. The API must keep backward compatibility as long as the major version is not updated.

To reach the actual low-level graphics API, graphics operations go through several layers described below.

ui.Image

An Image in this package is an intermediate image to connect the upper level (ebiten.Image) and a lower level (mipmap.Mipmap). This image caches some values from ebiten.Image calls for more efficient renderings.

(TBD)

mipmap.Mipmap

A Mipmap in the package mipmap represents a collection of images for mipmap. With linear-filter, prepared shrunk images are needed for high quality rendering. A mipmap has an original image, a half size, 1/4, 1/8, … and so on. They are created when the original image is used as a rendering source with linear-filter. They are discarded when an original image is used as a rendering destination.

For more details about mipmap, see https://en.wikipedia.org/wiki/Mipmap.

buffered.Image

An Image has pixel cache, which is updated when ReadPixels is called, in order to reduce the graphics commands to read pixels.

atlas.Image and atlas.backend

An image in the package atlas represents an image that is a part of a big image (backend). The purpose is to achieve more efficient rendering. In general, the smaller the number of graphics commands is, the better the performance will be. If successive two graphics operations have different source images, they have to be executed as different graphics commands. This is not efficient. If they have the same source image, the operations can be merged into one graphics command. To achieve this, images are integrated into one big image whenever possible.

The packing algorithm is a very simple binary tree, which splits a region into two horizontally or vertically. See github.com/hajimehoshi/ebiten/v2/internal/packing for more details.

A destination image for rendering cannot be put into a big image as a part. In the current implementation, such an orphan image is put into a big image again after the image is used as a source a number of times. On the other hand, if an image that is on a big image is used as a rendering destination, the image will be an orphan image automatically before rendering.

graphicscommand.Image

An image in the package graphicscommand represents an interface of graphics operations. To make rendering efficient, graphics operations are merged when possible. For example, if a source image and a destination image in successive operations are the same, the operations might be merged into one graphics command. To be exact, there are more complex conditions to merge operations. For details, see https://pkg.go.dev/github.com/hajimehoshi/ebiten/v2#Image.DrawImage.

driver.Image

An image in the package driver is an interface for a low-level graphics driver like OpenGL or Metal. This executes the low-level APIs almost directly.

The package driver offers interfaces, not implementations. One of the implementations is in github.com/hajimehoshi/ebiten/v2/internal/graphicsdriver/opengl.

The low-level graphics driver basically does these things:

1. Set vertices and their indices.
2. Draw the vertices (Loop)
    a. Set a renderer source
    b. Set a renderer target
    c. Set the range of indices
    d. Set other properties (e.g., filter)
    e. Draw the specified vertices