Design Doc for Shape Detection Web API

This Document is Public

Authors: xianglu@chromium.org, mcasas@chromium.org (TL: reillyg@chromium.org)
Status: Draft
Last Updated: 2018-09-13
*Link: **http://tinyurl.com/shape-detection-in-chromium***

Objective

Implement the WICG Shape Detection API on platforms with operating system libraries supporting it:

Detector	Android	macOS	Windows 10	Linux	Chrome OS
Face	✓	✓‡	✓	✗	✗
Barcode	✓†	✓	✗	✗	✗
Text	✓†	✓‡	✓	✗	✗

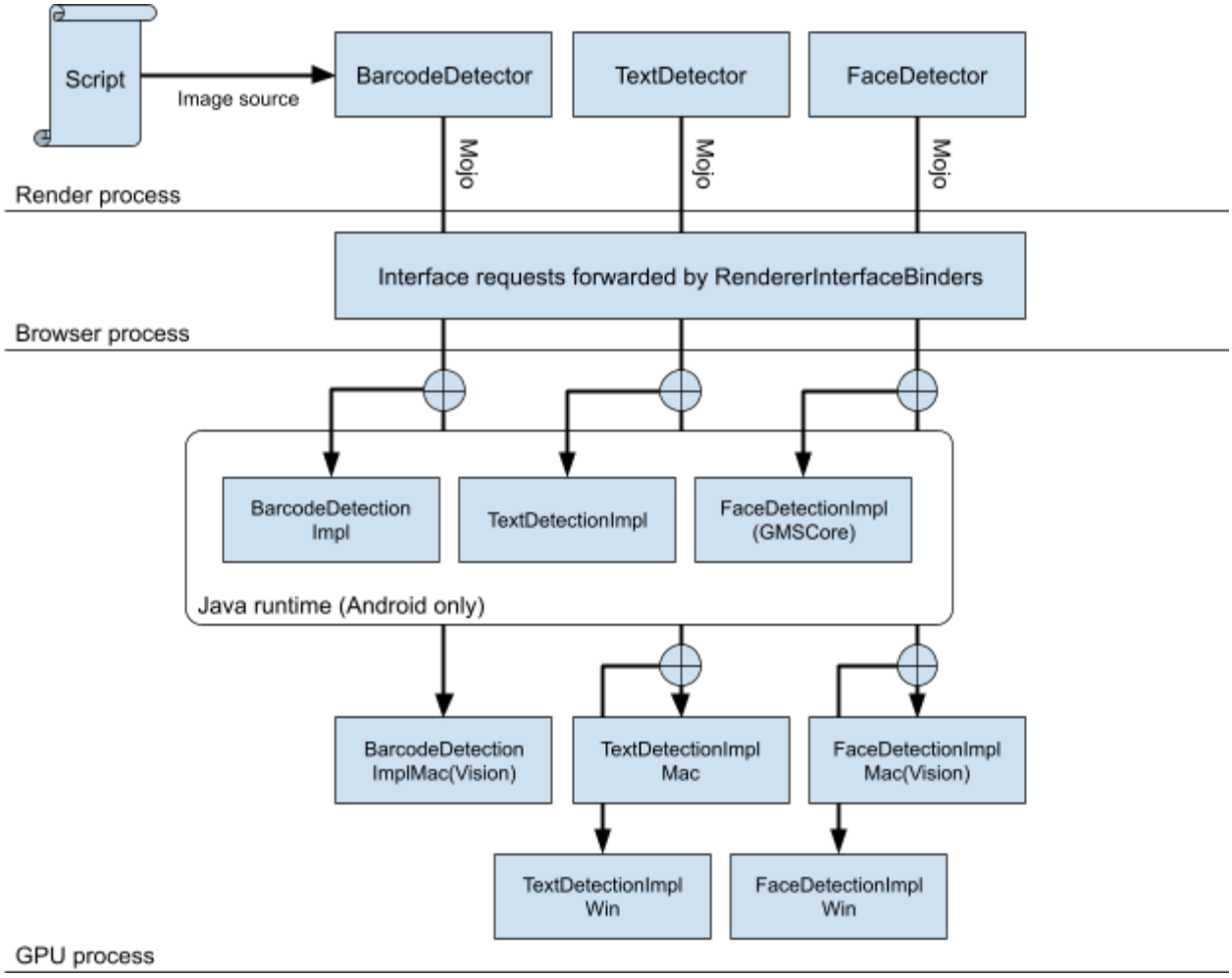
†Requires the Play support libraries.
‡Improved accuracy using the Vision Framework in macOS 10.13 and later.

Background

Photos and images constitute the largest chunk of the Web, and many include recognisable features, such as human faces or QR codes. Detecting these features is computationally expensive, but would lead to interesting use cases e.g. face tagging or detection of high saliency areas. Also, users interacting with WebCams or other Video Capture Devices have become accustomed to camera-like features such as the ability to focus directly on human faces on the screen of their devices. This is particularly true in the case of mobile devices, where hardware manufacturers have long been supporting these features. Unfortunately, Web Apps do not yet have access to these hardware capabilities, which makes the use of computationally demanding libraries necessary.

Overview

End users should be able to call detect() on an source of images. Blink will extract the raw image data and use Mojo to call out to platform dependent APIs for analysis.



Detailed Design

Blink

The web-exposed classes, `BarcodeDetector`, `FaceDetector` and `TextDetector` all inherit from a common (internal) `ShapeDetector` class. They take an `ImageBitmapSourceUnion` from the web, retrieve the raw image data as an `SkBitmap` and send a request for detection through appropriate Mojo interface.

The render process requests connections to the shape detection Mojo interfaces from the browser process, which forwards those requests to the Shape Detection service hosted in the GPU process.

Android

The Shape Detection service on Android is implemented in Java and runs in the GPU process. If the Play support library is available then the detectors are implemented using the com.google.android.gms.vision package's BarcodeDetector, FaceDetector and TextRecognizer

classes. Without this library only face detection is available using the built-in android.media.FaceDetector classes.

It is technically possible to call into some of these platform APIs for shape detection directly from an Android render process via JNI (see this CL) because there is a Java runtime available there. However, but peter@ and palmer@ argued convincingly that we should not take this approach because:

- The Java runtime in the render process is (mostly?) unused and so we would like to save RAM (enabling security features like Site Isolation) by disabling it.
- Calling these APIs may introduce dependencies on external system resources that would foil later efforts to further sandbox the render process.

macOS

On macOS the service is implemented in C++ in the GPU process. On versions prior to macOS High Sierra 10.13 the CIDetector API is used for all detector types. This limits barcode detection to QR codes only. On High Sierra and later the Vision Framework is used for face and text detection which improves both accuracy and the number of barcode symbologies supported. These APIs are implemented by the system frameworks using GPU acceleration.

Windows

On Windows the service is also implemented in C++ in the GPU process. Face detection is implemented with using the Windows.Media.FaceAnalysis.FaceDetector class and text detection with Windows.Media.Ocr.OcrEngine.

Project Information

- Spec (Editor: mcasas): https://wicg.github.io/shape-detection-api/
- Bugs: 646035, 659138, bug list
- Blink implementation: //src/third_party/blink/renderer/modules/shapedetection/
- Android implementation: //src/services/shape_detection/android/java/...
- macOS/Windows implementation: //src/services/shape_detection/

Risks and alternatives

This API is risky because it is exercising additional system interfaces which may be unstable and decrease the stability of Chrome or the operating system in general. The use of GPU acceleration in particular has been flagged as a potential risk as it may expose bugs in GPU drivers and hardware that are very system-specific.

The alternative to this API is the status quo, where sites must implement their own image processing routines using either JavaScript or WebAssembly. These will necessarily be lower

performance because they cannot take advantage of hardware acceleration (at least until WebGL 2.0 Compute is available).

Core Principle Considerations

Speed (and System Health)

Shape detection is performed asynchronously (so it does not block the renderer main thread). The image data is passed between processes using shared memory. In the GPU process the detection task itself is asynchronous.

Stability

Detection takes place inside the GPU process which is restartable in case of crashes. We have a feature request out to the Origin Trials team to flag GPU process crashes in browser sessions in which these APIs have been used so we can determine if using these platform APIs decreases GPU process stability.

Security

Potentially unsafe image data is decoded by the render process and converted to an `SkBitmap` which is safe to pass to other process. Detection itself happens in the GPU process. Since the browser process connects the render process's request directly to the GPU process it processes neither the raw image nor the detection result.

Simplicity

This feature is invisible to users, providing only improved page performance.

Privacy Considerations

This API is an optimization of capabilities already available to web sites that ship their own object detection code. The privacy considerations for this API can be considered separately from those for the APIs from which a site gets access to the image data processed by this API, such as the camera.

This API will not be available on all platforms and may vary in capability due to hardware or operating system version. This constitutes a potential fingerprinting vector but is generally low entropy.

The API implementation leverages built-in OS features rather than external services so neither the raw image data nor the detection results are stored or transmitted by the browser.

Testing Plan

Content browser tests: //src/content/browser/shapedetection/shapedetection_browsertest.cc
Layout tests: //src/third_party/WebKit/LayoutTests/shapedetection/
//src/third_party/WebKit/LayoutTests/fast/shapedetection/
//src/third_party/WebKit/LayoutTests/test/http/shapedetection/