Dissertation Type: enterprise

DEPARTMENT OF COMPUTER SCIENCE

Implementing CSS conic gradients in the Firefox rendering engine

Tim Nguyen

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Friday 15th May, 2020

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MEng in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Tim Nguyen, Friday 15th May, 2020

Contents

List of Figures

Executive Summary

With the web being increasingly used for modern applications, having powerful web technologies is more and more important nowadays. This project aims to implement CSS conic gradients into Firefox, the second most used desktop web browser, open source and developed by Mozilla.

According to 'Can I use' [1], conic gradients were implemented and enabled by default in:

- Google Chrome 69 (released in September 2018)
- Microsoft Edge since it switched to Chromium (in January 2020)
- Safari 12.1 (released in March 2019)

With the complexity of browser rendering engines and of the Firefox development process in mind, making an implementation of a web standard that is interoperable with other web browsers is not an easy task, since it requires extensive programming, testing and collaboration.

Despite those challenges, conic gradients have been implemented for most platforms (macOS, Linux, some Windows devices and Android) behind a feature flag in Firefox 75, released on April 8th. Once the feature flag is enabled, web developers will be able to use conic gradients more widely without a polyfill and Mozilla will be able to catch up to the competition.

Supporting Technologies

- I used parts of pre-existing code written for the linear and radial gradient implementations in Gecko and altered it to support parts of my implementation.
- I used Mozilla's infrastructure for parts of the Firefox development process.
- A Skia library method was used to support the Skia graphics backend implementation in Gecko.
- Some test cases and examples were inspired by examples on Lea Verou polyfill's website at https://leaverou.github.io/conic-gradient.

Acknowledgements

Credits go to the following people for making this project a success:

- The Firefox Layout Team in particular to Jonathan Watt and Sean Voisen for finding this project
- The Firefox Graphics Team for WebRender code reviews and answering questions related to WebRender
- Dr. David Bernhard for supervising this project
- Emilio Cobós Álvarez for most code reviews and answering questions related to the Firefox Style System
- Lea Verou for being behind the original specification and for making the very helpful conic gradient polyfill
- Lee Salzman for Skia code reviews
- Markus Stange and Matt Woodrow for web painting code reviews
- James Graham and Gabriel Luong for some other code reviews

Chapter 1

Background

Conic gradients, also known as angular gradients or sweep gradients, are gradients where color stops are placed at different angles around a circle. The resulting image looks like a cone, hence the name conic gradient. The `conic-gradient` function has been suggested to the CSS standards by Lea Verou [2], currently an HCI researcher at the MIT Computer Science & Artificial Intelligence Lab [3].

This project is about implementing this function on the Firefox rendering engine, Gecko, currently used on Android, Linux, macOS, Windows and for printers. Firefox for iOS is not fully relevant to this project given that all iOS web browsers are restricted to use the WebKit rendering engine, although this project has uncovered a bug in WebKit's implementation, which will be described later on.

To understand the usefulness of this project, it is important to first describe the challenges behind the conception of web technologies.

1.1 Anatomy of a web page

Web pages are documents written using markup languages like HTML. HTML is an XML-based language, where content is described via tags:

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello world</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <p>This is a paragraph</p>
</body>
</html>
```

Scripting for web pages can be done via JavaScript and associated to the HTML page via a `<script>` tag.

The Cascading Style Sheets (CSS) programming language can be used to define styling (background images, text colors, etc.) for documents written in HTML, XHTML, SVG or other markup languages typically used along with web pages.

1.1.1 CSS

A CSS file or stylesheet contains a set of CSS rules, which themselves contain a CSS selector defining which elements should be styled, and a set of CSS properties and values which defines how those elements should be styled.

For instance, the following CSS rule gives all paragraphs white text and a blue background:

```
p {
    background-color: blue;
    color: white;
}
```

CSS selectors have the concept of specificity which handle the case of conflicting rules: the rule which has the selector of the highest specificity will take precedence. This will not be defined in detail, since it is not highly relevant to this project.

CSS also comes with functions, like the `conic-gradient` function, which can be used as any image value:

```
div {
    background-image: conic-gradient(red, blue);
}
```

CSS was originally small enough that it evolved in versions:

1. CSS 1, published in 1996 [4]: It had support for basic alignment, color, layout and text-related properties.
2. CSS 2, published in 1998 [5]: It added support for layering via z-index, positioning, media queries, bidirectional text (for RTL languages) and shadows.
3. CSS 2.1, with the final publication done in 2010 [6]: It aimed to fix problems from CSS 2 by removing some features.

CSS was then split into multiple modules, like the "CSS Backgrounds and Borders Module" or the "CSS Fonts Module" which evolve via levels for which the process will be described in the next section. The version commonly referred to as "CSS 3" is in reality a set of modules, but such a version does not officially exist.

1.2 Web standards

Web standards are mainly done by two bodies:

- the WHATWG (Web Hypertext Application Technology Working Group), originally founded by individuals from Apple, Mozilla and Opera Software in 2004, in reaction to W3C's direction towards XHTML and lack of interest for HTML and web developer needs. Since then, individuals from Google or Microsoft have joined. [7]
- the W3C (World Wide Web Consortium), the original standard organisation behind web standards founded by Tim Berners-Lee, consisting of full-time staff and member organisations. [8]

After an agreement between the two bodies in 2019, the WHATWG maintains the HTML and DOM (Document Object Model) standards. Whereas the W3C maintains standards for CSS, SVG and most other web technologies. The W3C used to maintain their own version of the HTML standard before this agreement. [9]

For W3C standards like CSS, the process for which a first draft becomes a recommended standard goes as follows (Quoted from [10]):

1. Publication of the First Public Working Draft.
2. Publication of zero or more revised Working Drafts.
3. Publication of a Candidate Recommendation (CR).
4. Publication of a Proposed Recommendation (PR).
5. Publication as a W3C Recommendation (REC).
6. Possibly, Publication as an Edited or Amended Recommendation

When a specification reaches candidate recommendation stage, it is expected to be detailed and not require major changes. Browser vendors are recommended to implement and enable by default features from specifications only when the specification reaches at least candidate recommendation stage. In practice, it is not the case, since browser vendors often ship features from early drafts or not part of the standards. It can be done for various reasons:

- The feature has consensus, but is part of a bigger specification that would take a long time to become a recommendation.
- It is done to find issues that arise from implementing a certain feature, in which case it is recommended to put the feature behind a feature flag.
- Lobbying by a browser vendor.

At the time of writing, CSS conic gradients are part of the "CSS Image Values and Replaced Content Module Level 4" specification [11] which is a working draft. According to 'Can I use' [1], they were implemented and enabled by default in:

- Google Chrome 69 (released in September 2018)
- Microsoft Edge since it switched to Chromium (in January 2020)
- Safari 12.1 (released in March 2019)

They are currently unsupported in Internet Explorer and some other minor browsers. Before this project, they were also completely unimplemented in Mozilla Firefox. This means that it is a web compatibility issue for websites using them.

It may be worth wondering why switching to Chromium is not being done by Mozilla when Microsoft and Opera have done it, as it would give Firefox the technology for free.

With only three major rendering engines left:

- WebKit: the rendering engine used by Safari and all web browsers on iOS
- Blink: the rendering engine used by Chromium-based browsers (Chrome, Opera, Microsoft Edge)
- Gecko: Firefox's rendering engine

Diversity of rendering engines is important to prevent excessive lobbying by a single company and also helps uncover bugs in other implementations of web standards through collaboration.

1.3 Web compatibility

1.3.1 Lack of support

Imagine multiple bottle formats, each bottle having different bottle caps. If someone buys a certain bottle, they would need the appropriate bottle opener to open the bottle.

One type of web compatibility issue is similar: if a website (the bottle) uses a technology (the bottle cap) that is only supported by one web browser (the bottle opener), the website will not be usable by every browser.

This type of issue is usually not common though, given most websites will only use technologies that are supported by most web browsers, to avoid losing out on visitors. The same is applicable to the bottle opener example, it is fairly uncommon to encounter unusual bottle caps requiring special bottle openers, as such a bottle brand would lose out on customers.

Polyfills exist to cope with this problem on websites. They act as a compatibility layer, allowing websites to use a newer technology while delivering the same content to all visitors, through older but more complex technologies.

The main problem with polyfills is that they are under the form of a JavaScript file, taking up network bandwidth when visiting production websites. JavaScript is also occasionally disabled for security reasons [12]. Polyfills have relatively low usage for those reasons. This justifies the usefulness of implementing the newer technology directly in all web browsers.

CSS conic gradients belong in this category of technologies with respect to the Firefox rendering engine.

1.3.2 Incompatibilities

Different forms of Badminton have evolved differently across Eurasia [13]. However, only one form has been standardised across the world for competitions. This allows avoiding conflicting shuttlecocks or courts. This does mean that some older courts may not be usable for competitions, leading to incompatibility.

The web had similar issues when web technologies were not well-standardised yet. Quirks mode in HTML is an example of technology that needs to be implemented by all browser engines nowadays to support older websites. It is a set of non-standard behaviours that Microsoft Internet Explorer 5 and Netscape Navigator 4 used to have with HTML [14]. Since then, it has its own specification, which is a living standard [15].

However, browser wars have prevented Microsoft from taking the monopoly on the browser market, when Firefox started gaining a significant usage share around 2009 and slowly killing Microsoft Internet Explorer. Although Mozilla was relatively successful democratising web standards, it was never able to win over the majority of the market [16], due to the arrival of WebKit and Blink: WebKit was Chrome's and Safari's engine, while Blink was Google's fork of WebKit around 2013.

With WebKit then Blink's domination, web compatibility issues continued arising, this time due to a different pattern. After the browser wars, while standards started evolving in distinct steps as defined in the previous section, these steps were relatively new, which lead to different problems, one of them being the following pattern:

1. Browser A implements technology 1
2. Browser B implements technology 2, a newer version of technology 1
3. Website I uses technology 1
4. Website II uses technology 2
5. Result: Website I only works in browser A, Website II only works in browser B

CSS linear and radial gradients are examples of such technologies. The first prefixed form was introduced in 2008 [17]:

```
-webkit-gradient(linear, left top, left bottom, color-stop(0%, red), color-stop(100%, blue));
```

The second form was introduced in 2011, with the goal of having an easier syntax to use [18]:

```
-webkit-linear-gradient(top, red, blue);
```

The unprefixed version was then finally introduced [19] (notice the slightly different first argument):

```
linear-gradient(to bottom, red, blue);
```

These technologies were originally implemented with prefixes to allow implementing drafts while being able to change them when they evolve into recommendations. Since the technologies were enabled by default, website production code started using the prefixed forms, making it necessary to keep supporting them for compatibility.

Conic gradients are part of a level 4 specification, which, like other new standards, takes in account web compatibility problems in the process by forcing the implementation to be done behind an internal preference until fully ready, instead of using prefixes, to prevent improper widespread use in website production code.

1.4 Pre-existing gradient types

Since a lot of the implementation is based on the pre-existing gradient types: linear and radial gradients. It is worth quickly mentioning what they are in this section.

The `-webkit-` and `-moz-` prefixed forms with a slightly different syntax are currently supported by Firefox. In fact, the `-webkit-` prefix for those two gradient types, is actually officially part of WHATWG's web compatibility standard [20].

1.4.1 Linear gradients

```
background-image: linear-gradient(red, gold);
/* equivalent to: */
background-image: linear-gradient(red 0%, gold 100%);
/* or: */
background-image: linear-gradient(to bottom, red, gold);
/* or: */
background-image: -webkit-linear-gradient(top, red, gold);
```

1.4.2 Radial gradients

```
background-image: radial-gradient(red, gold);
/* equivalent to: */
background-image: radial-gradient(red 0%, gold 100%);
/* or: */
background-image: radial-gradient(at center, red, gold);
/* or: */
background-image: -webkit-radial-gradient(center, red, gold);
```

1.5 Conic gradients

Conic gradients have been popular in the web development and web design community as some blog posts show [21] [22] [2]. The popularity can also be seen through a Twitter search for "conic gradient" [23], showcasing many beautiful CSS demos from the community.

The main reason it has not been widely used is due to the lack of support in Mozilla Firefox, which many of the blog posts and tweets advocate for. Despite that last fact, the polyfill [24] created by Lea Verou, who originally suggested the standard, has some popularity which can be seen with a non-exhaustive Github search [25].

1.5.1 Syntax

The full syntax for conic gradients can be found in appendix A, but can be summarised as:

```
conic-gradient(
  [ from <angle> ]? [ at <position> ]?,
  <angular-color-stop-list>
)
```

The function allows specifying the starting angle after the `from` keyword, where the zero angle is pointing upwards similar to a clock. The center can also be specified after the `at` keyword. Those two first parameters are optional, where the angle defaults to `0deg` and the center defaults to `50% 50%`.

The following figure, extracted from the specification [26], illustrates how `conic-gradient(at 25% 30%, white, black 60%)` renders:

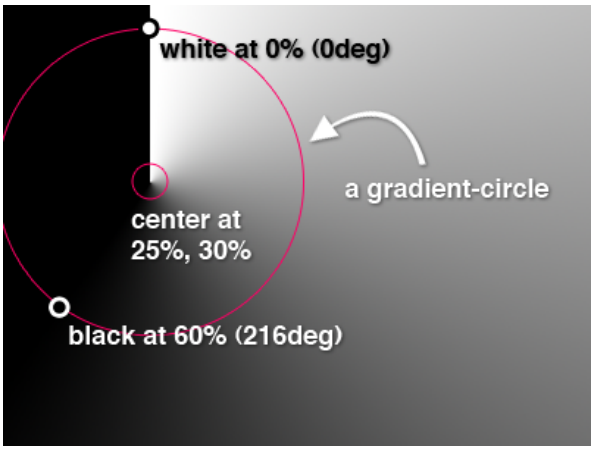


Figure 1.1: Example diagram of a conic-gradient

After the angle and center parameters, follows the color stop list. Similarly to `linear-gradient` and `radial-gradient`, each color stop can take:

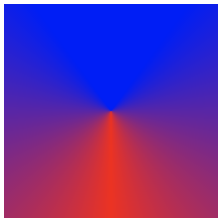
- no position, where the position is interpolated implicitly
- one position, where the position is as specified
- two positions, where the stop is equivalent to specifying two stops with the same color, but at two different positions

The color stop positions are called interpolation hints or offsets. The way stops are interpolated with each other is similar to other gradients and is defined in appendix B. Unlike the other gradient types, the positions are angles or percentages, as opposed to lengths or percentages. Unless it is the first stop, color stops may omit the color, in which case it corresponds to the one from the previous stop.

Here are some more examples to illustrate how the arguments work:

```
background-image: conic-gradient(red, gold);
/* equivalent to: */
background-image: conic-gradient(red 0%, gold 100%);
/* or: */
background-image: conic-gradient(red 0% 0%, gold 100% 100%);
/* or: */
background-image: conic-gradient(red 0%, 0%, gold);
```

Figure 1.2: Simple example



```
background-image: conic-gradient(blue .1turn, red, blue .9turn);
/* equivalent to: */
background-image: conic-gradient(blue 10%, red 50%, blue 90%);
/* or: */
background-image: conic-gradient(blue 0% 10%, red 50% 50%, blue 90% 100%);
```

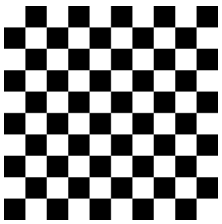
Figure 1.3: Cone example

When combined with other CSS features, it is possible to do more nice demos. The following two examples are taken from the polyfill's website [24].

```
background-image: conic-gradient(red, yellow, lime, aqua, blue, magenta, red);
border-radius: 50%;
```

Figure 1.4: Color wheel example

Since background images repeat by default, a checkerboard can be created by scaling down the background size:



```
background-size: 20% 20%;
background-image: conic-gradient(black 25%, white 0 50%, black 0 75%, white 0);
/* equivalent to: */
background-image: conic-gradient(black 25%, white 25% 50%, black 50% 75%, white 75%);
```

Figure 1.5: Checkerboard example

In the previous example, the first `2×2` square is the original gradient, but it is repeated across the element. This is not to be mistaken with the `repeating-conic-gradient` function described in the next section.

1.5.2 Gradients with repeating stops

A gradient with infinitely repeating stops can be specified using the `repeating-conic-gradient` function sharing the same syntax as `conic-gradient`. This function is analogous to `repeating-linear-gradient` or `repeating-radial-gradient`.

The following example creates a starburst effect:

```
background-image: repeating-conic-gradient(black 0 15deg, white 15deg 30deg);
```

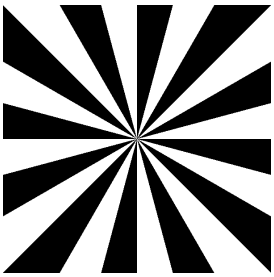


Figure 1.6: repeating-conic-gradient example

This prevents the need for repeating the black and white stops over and over again, as it would be needed with the `conic-gradient` function:

```
background-image: conic-gradient(
    black 0 15deg,
    white 15deg 30deg,
    black 30deg 45deg,
    white 45deg 60deg,
    ...
    black 330deg 345deg,
```

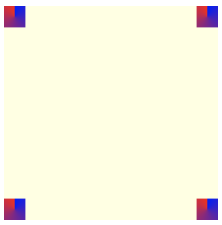
```
        white 345deg 360deg,
    );
```

1.5.3 Other uses

All of the examples above use the `background-image` property, since it is the most common use case, but conic gradients can be used instead of any CSS `<image>` value. This section will mention some other CSS properties that can also take image values.

border-image

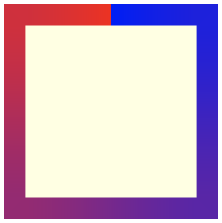
The image specified by the `border-image-source` CSS property is rendered within the border specified around the element.



```
border: 30px solid;
border-image-source: conic-gradient(blue, red);
background-color: lightyellow;
```

Figure 1.7: Basic `border-image` example

It is also possible to specify how the border image should be sliced:



```
border: 30px solid;
border-image-source: conic-gradient(blue, red);
border-image-slice: 1;
background-color: lightyellow;
```

Figure 1.8: `border-image-slice` example

mask-image

The mask-image property applies the image as an alpha mask on the element. Areas where the mask is opaque will be rendered, whereas areas where the mask is transparent will be masked.

```
mask-image: conic-gradient(black, transparent);
background-image: linear-gradient(red, gold);
```

Figure 1.9: `mask-image` example

Chapter 2

The Firefox Development process

This chapter describes how to set up a development environment to build Firefox and the Firefox development process in general. It is worth noting that due to the 20 year-old history of the source code, there are different workflows that have been developed and documented which all work today. This section will describe the workflow that I have used for development, which should be close to the latest recommended one.

2.1 Finding issues to work on

While some of Mozilla's projects are on Github, most older projects like Firefox are tracked on Mozilla's own issue tracker called Bugzilla [27].

To find simple issues to get familiar with the development process, there are "Good first bugs" which can be browsed on Codetribute [28]. These issues usually have mentors assigned to them and are simple changes with step-by-step instructions.

2.2 Getting the source code

Firefox development uses Mercurial (hg) [29], a version-control system similar to Git, although Git can also be used through ports or mirrors.

The command mentioned in the documentation to clone the Firefox source code is:

```
hg clone --uncompressed https://hg.mozilla.org/mozilla-unified
```

mozilla-unified is the repository combining the commit history of all other Firefox repositories like mozilla-central, mozilla-beta or mozilla-release (which will be described at the end of this chapter) as different branches.

The command above usually takes about 30 minutes depending on the network connection. Once the cloning is done, the central branch should be checked out, since it contains the latest stable changes:

```
hg up central
```

`hg up` is a shortcut for `hg update`, which is equivalent to the `git checkout` command on Git, as it lets you check out the source code at different commits.

2.3 Installing dependencies

Once the source code is cloned, the dependencies can be installed by running the following command [30]:

```
./mach bootstrap
```

This will bring up an interactive wizard to install the tools appropriate for the developer. The wizard will ask whether to use full or artifact builds at some point. Full builds are used for C++ and Rust changes,

since they require the full binaries to be regenerated, while artifact builds are typically used for frontend changes where binaries can be downloaded from a server [30]. For this project, full builds are necessary since the project involves changes on the rendering engine written in C++ and Rust.

On Windows, extra steps need to be done beforehand, as mentioned on the documentation [31]:

1. You need a 64-bit version of Windows 7 or later.
2. Download and install Visual Studio.
3. Finally download the MozillaBuild Package. Installation directory should be: `c:\\mozilla-build\`

2.4 Building Firefox

Once the changes have been made to the source code, in order to test the changes, building Firefox is needed. This is done using the following command:

```
./mach build
```

The first time, this command takes about one hour for full builds depending on the hardware [30], while artifact builds take about 5 minutes depending on the network connection. Once finished, the build can be ran using:

```
./mach run
```

2.5 Getting around the source code

Mozilla provides a very useful search tool called Searchfox [32] where code is regularly indexed from different Mozilla repositories. In search results, Searchfox can distinguish between function definitions, function declarations (for C++), function calls and raw text matches. Although not showcased in the screenshot, it also displays results from test files separately for convenience and build-generated files can be also be searched through.

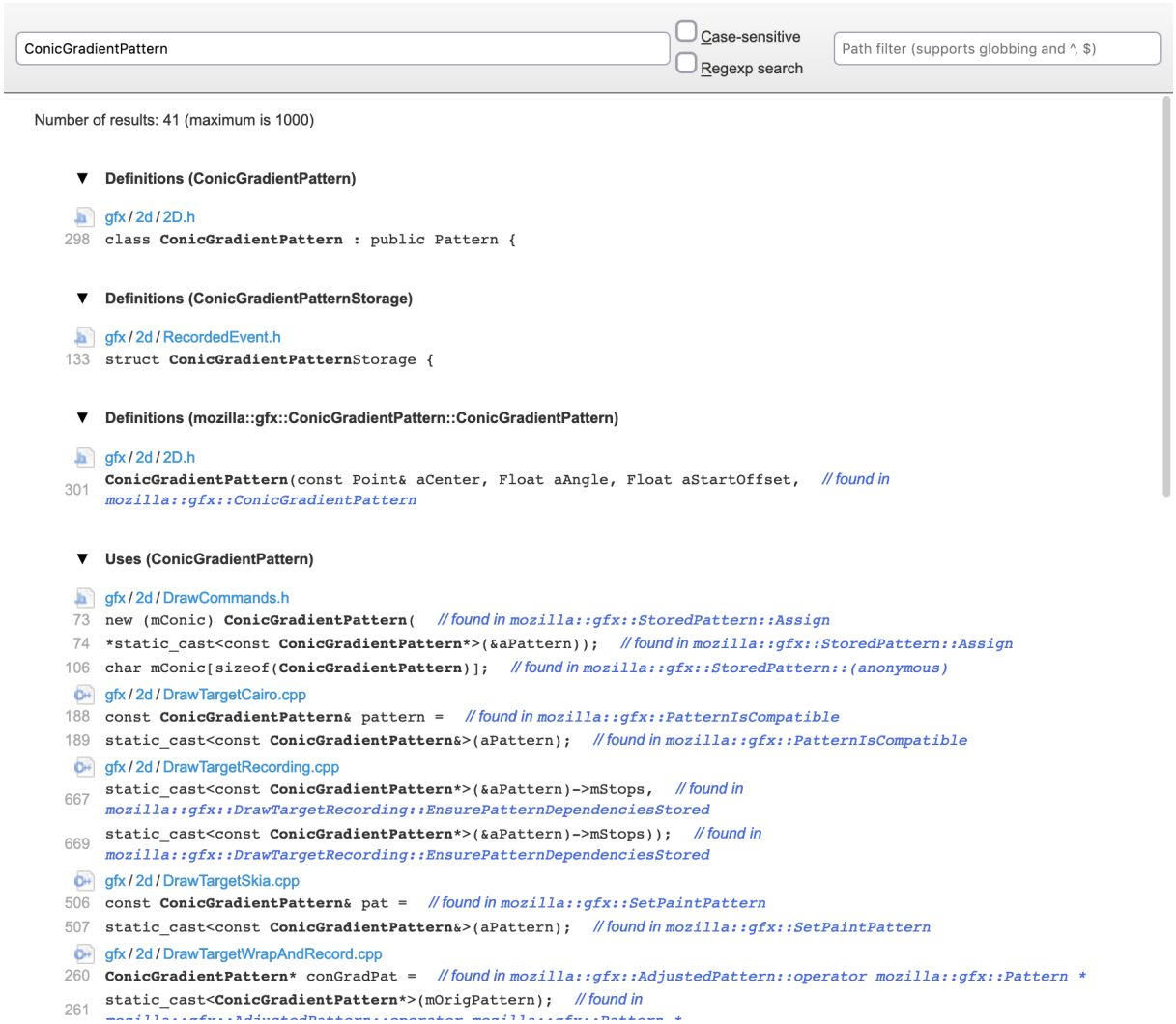


Figure 2.1: Screenshot of a Searchfox search for ConicGradientPattern

In the code view, there is an annotate (or blame) sidebar, which shows which commit added or last changed a certain line of code, and code can be browsed through different points of history if needed. The code view also provides different links:

- a link to file a Bugzilla issue in the component relevant to the file,
- a link to the log showing all commits that have affected this file,
- a link to the raw file,
- links to open the file in other tools (HG Web, Code Coverage, DXR).

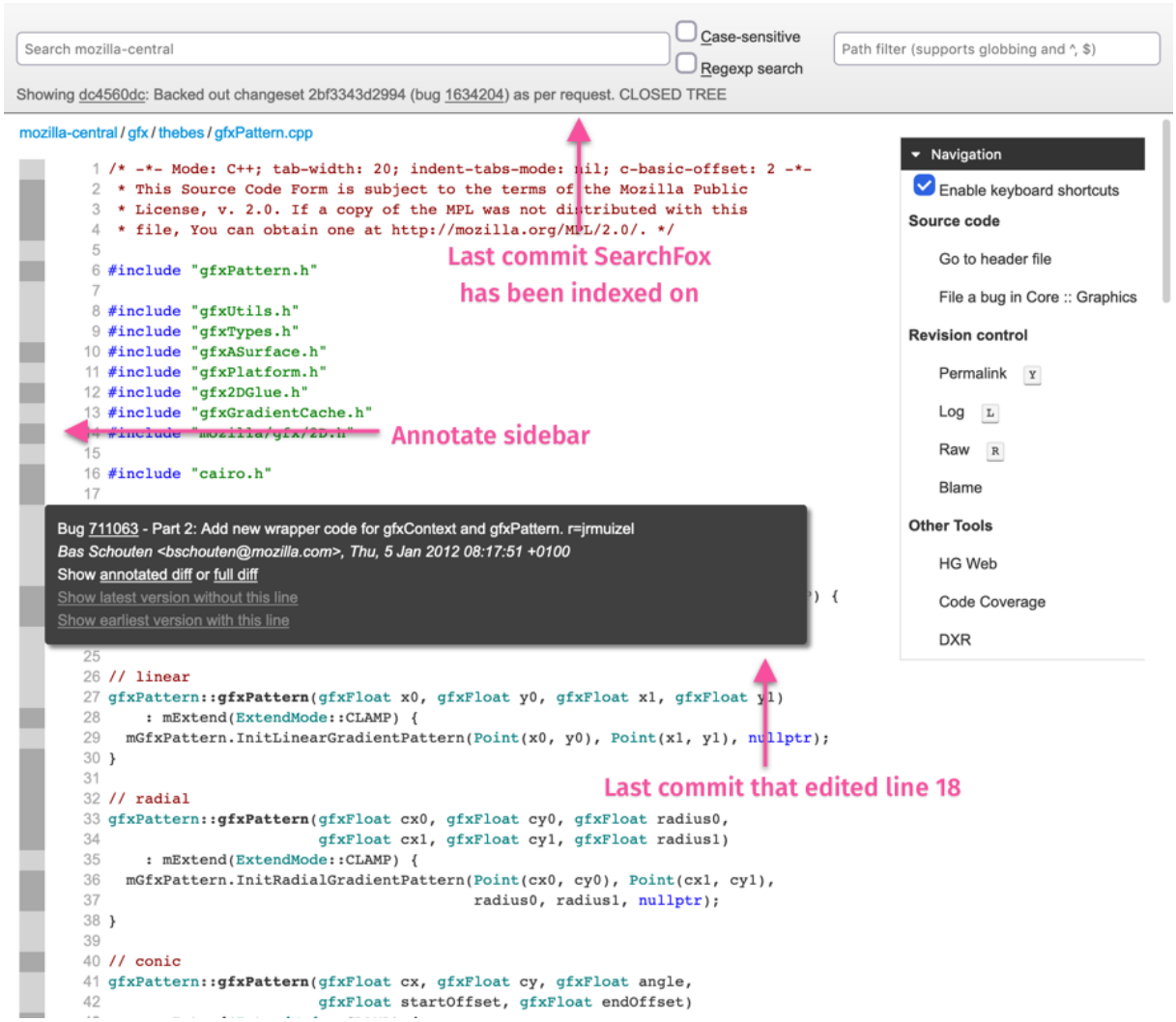


Figure 2.2: Searchfox file view for `gfx/thebes/gfxPattern.cpp`

The log link and the annotate sidebar are particularly useful to understand why a piece of code was written in a certain way, especially since both of these contain references to the associated Bugzilla issues where comments from the author or the reviewer can be found.

2.6 Committing the changes

Once the changes have been are tested, they should be committed to be sent for review. This is done using the command below:

```
hg commit -m "Bug XXX - Commit message. r=reviewer"
```

where:

- `XXX` is the issue number on Bugzilla.
- "Commit message" is a sentence summarising the changes.
- the reviewer is referred to by their nickname.

Subsequent changes to the same revision can be done by amending the commit:

```
hg commit --amend
```

If the commit is out of date, it is possible to update it by pulling the latest source code using `hg pull`, then by rebasing the commit on top of the central branch using `hg rebase -d central`.

2.7 Submitting for review

Mozilla uses a third-party tool called Phabricator [33] for code reviews. This process is analogous to pull requests on Github. Mozilla's instance of Phabricator [34] is integrated with a set of in-house tools:

- `moz-phab`: a command-line utility to publish commits to Phabricator.
- Bugzilla integration bot, linking Bugzilla issues with Phabricator revisions.
- A code review bot that runs on every submitted revision and adding automated. review comments from linters.
- Lando: a tool that allows merging the revision once it is accepted.

In order to submit a commit for review, it is necessary to have a Bugzilla account to login to Phabricator. Once logged into Phabricator, assuming the commit to be submitted is already checked out, submitting for review can be done using the `moz-phab submit` command.

The commit is now submitted for review. Subsequent changes to address comments from reviewers should be amended to the commit using `hg commit --amend` and re-submitted using `moz-phab submit`.

2.8 Testing

To ensure changes work correctly and do not break existing features, it is necessary to perform testing on the software. Manual testing can be done by the developer by running the compiled build of Firefox. However, for complex software like Firefox, there are a lot of tasks that need to be tested, so manual testing would take a large amount of time. This is where automated tests become useful.

There are many types of tests in Firefox, but for the sake of simplicity, only the ones relevant to this project will be described.

2.8.1 Mochitests

Quoted from the "Mochitest" Mozilla Developer Network page [35]:

> Mochitest is an automated testing framework built on top of the MochiKit JavaScript libraries. It is an automated regression testing framework used by Mozilla to report success or failure to the test harness using JavaScript function calls.
>
> Mochitest's use of JavaScript function calls to communicate test success or failure can be unsuitable for certain types of test. Only things that can be tested using JavaScript (with privileges!) can be tested with this framework. Given some creativity, that's actually much more than you might first think, but it's not possible to write Mochitest tests to directly test a non-scripted C++ component, for example. (Use a compiled-code test to do that.)

The main command to run mochitests is simply:

```
./mach mochitest
```

This runs all the mochitests, taking a big amount of time, but using different parameters, it can run only a subset of tests if needed.

No new mochitests will be added in this project, only pre-existing ones will be edited to add new test cases (usually one or two lines) to cover conic gradients.

2.8.2 Reference testing

Reference tests, also referred to as reftests, are automated tests that ensure the rendering of a certain testcase is correct. A reference test is composed of two files: the test and the reference. Both files contain code that should render the same way. The test file uses the feature that is being tested, while the reference file does not. If both files render the same, the test passes. If not, the test fails [36]. However, it is possible to allow a certain amount of difference by specifying the maximum number of different pixels and the maximum color channel (red, green or blue) difference in the test metadata. This is usually done when the difference is not directly due to the feature being tested, like a difference in how the graphics card handles both files [37].

When a test fails, the test log will include links to a screenshot of both the reference and the test files. They can be compared by pasting the logs into the reftest analyser tool [38], which can highlight where the pixels differ.

The following example is the reftest analyser highlighting differences between a test rendered locally and a png reference rendered on different hardware. There are 220 differing pixels, but differences are only by 1 in the color channels.

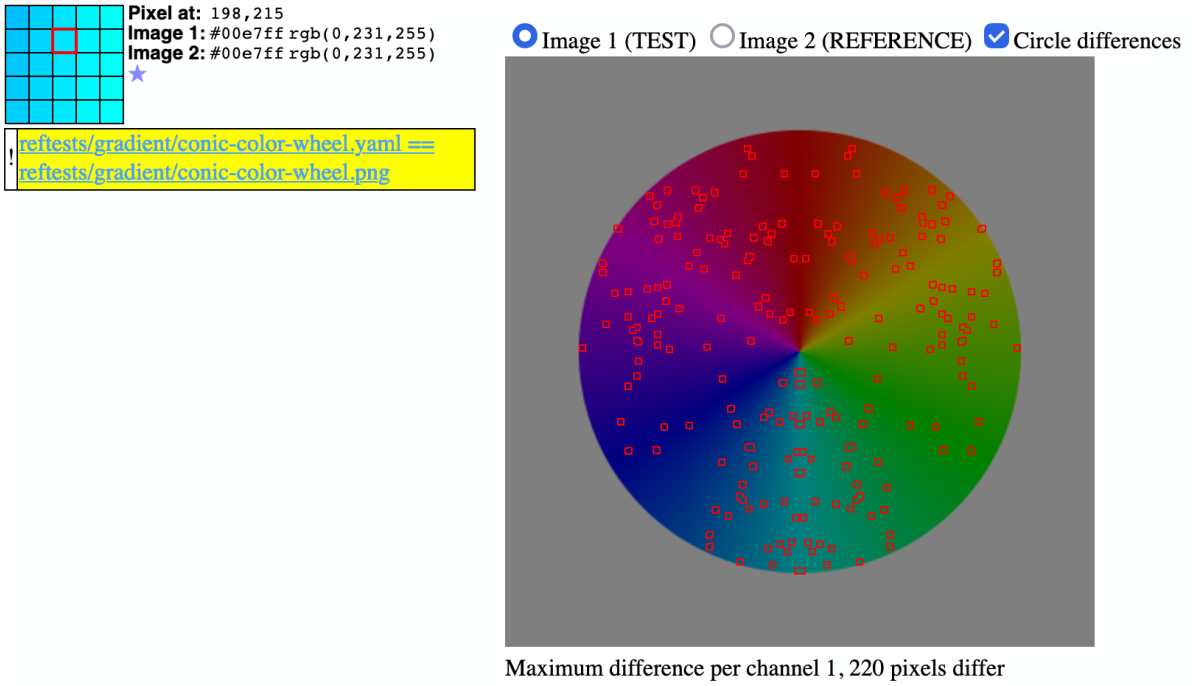


Figure 2.3: Reftest analyser

The `./mach reftest` command can run most types of reftests, but WebRender reftests need to be run with `cargo run -- -p 1 reftest` since they use different toolchains.

Reference testing is central to this project since most of the testing that will be added is visual. This project will add two types of reftests: Wrench and Web Platform Tests reference tests, which will be described in their respective sections.

2.8.3 The Try server

While it is possible to run tests locally, it is not possible to perform other tasks on the computer while waiting, since some tests require the test window to stay focused all the time. There are also many tests and many different supported configurations (different platforms, preferences, etc.), that it would take hours to run everything. To make it easier to execute entire test suites, Mozilla provides a "Try" server, where commits can be pushed as often as wanted and where tests will be ran on Mozilla's machines [39]. To use it, level 1 commit access must be requested, by filing an issue with the SSH public key attached, then by finding someone to vouch for access (usually the mentor of the task) [40].

Once level 1 commit access is acquired, it is possible to use the following command:

```
./mach try chooser
```

This will bring up a web page:

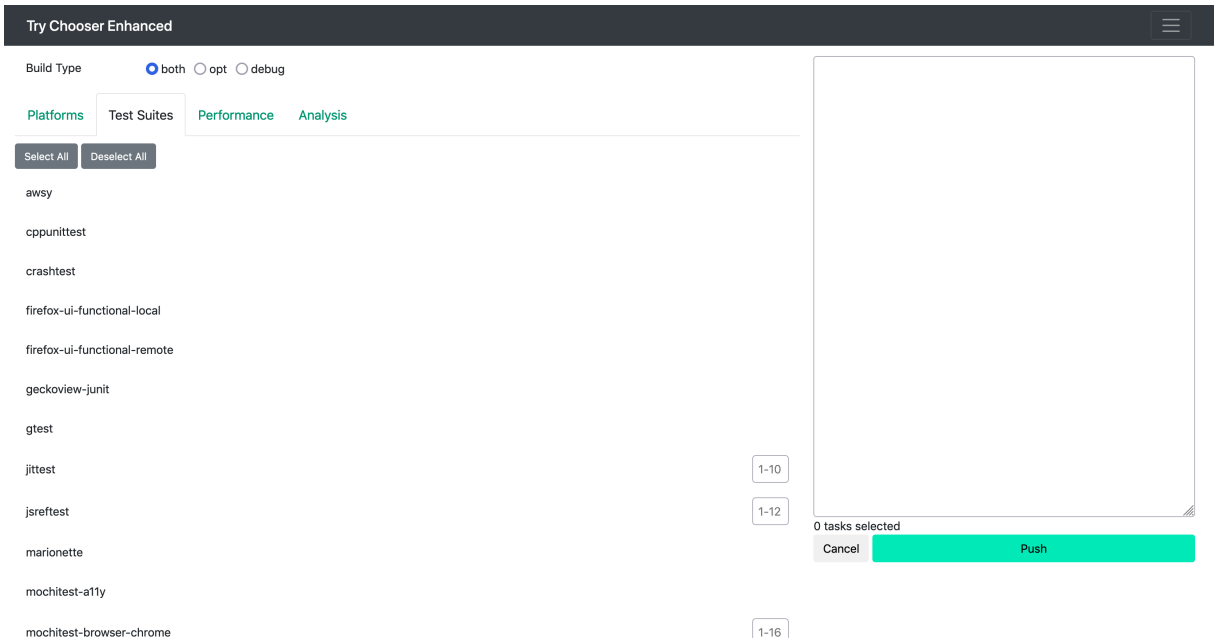


Figure 2.4: Try chooser

Different jobs/tasks/test suites and platforms can be selected and then pushed. A link will be provided in the command output to a tool called "Treeherder" [41] where test results can be checked after they are finished running. Here is a screenshot of the tool:

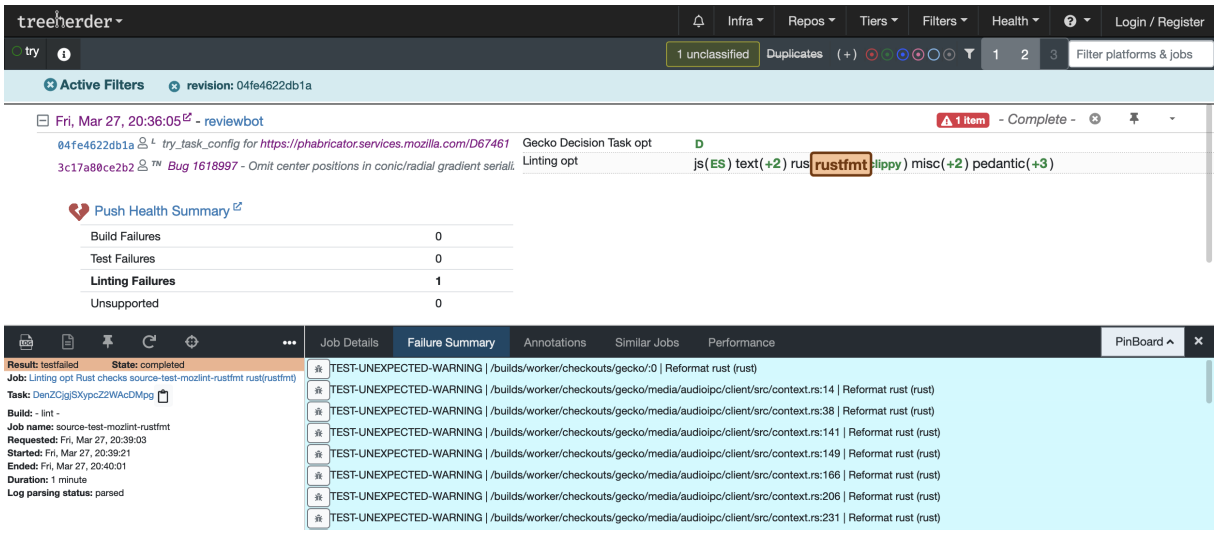


Figure 2.5: Treeherder

From the top, the first toolbar is the navigation bar, which contains links irrelevant to the try push. On the second toolbar, the following elements can be seen:

- The left side has the name of the repository ("try" in this case, since it is the try server where the commit is being pushed to)
- From left to right, buttons in the right side filter different jobs based on the status (fail, success, etc.), the importance or the platform or job names.

In the content area, the left side has the commits being pushed, while the right side displays the different jobs, color-coded based on the status. When a job is selected ("rustfmt" in this case), the bottom pane shows up with more details about it:

- The left sidebar has a toolbar with different actions to perform, relevant ones being "view logs" and "re-trigger job" (which may be useful to check whether a failure is intermittent or not). The job metadata is displayed under the toolbar.
- The right pane displays the logs, giving clues on why the job is failing.

2.9 Merging the change

When the revision is approved, the commit can be queued for check-in, using the "Check-in Needed" tag on Phabricator. The commit is then merged to the integration branch called "autoland", where all test suites run on Mozilla's machines. This process is analogous to continuous integration in other projects and results can be shown in the Treeherder UI decribed previously. If the commit causes a test failure, it is reverted from the integration branch. If there is no issue, the commit is merged along with other successful commits into the mozilla-central repository. The people taking care of manually doing the merges and reverting faulty commits are called "sheriffs".

mozilla-central contains the latest source code that is built twice a day by Mozilla's integration server into Firefox Nightly [42], the alpha version of Firefox, equivalent to Google Chrome Canary.

Every 4 weeks (previously 6 to 8 weeks) following a release calendar [43], the source code from mozilla-central is merged to mozilla-beta (the source code for Firefox Beta), and mozilla-beta is merged to mozilla-release (the source code for Firefox). After this merge, the version number is incremented for all 3 repositories. This work is done by the sheriffs, the release engineering and release management teams. There is a 4-day period before that merge date, called the soft freeze, where risky changes should be avoided and held for the next release.

For enterprises, Firefox has a yearly release called "extended supported release" (ESR) which is a version that only receives security updates for one year. It is in the mozilla-esrXX repository, where XX is the version number, and is merged from mozilla-release every year [43].

Sometimes, changes can be merged directly to the mozilla-beta, mozilla-release, mozilla-esrXX repositories if they are critical, with approval from release managers.

Chapter 3

Technical background

3.1 C++ and Rust

There are two programming languages used by the Firefox rendering engine, Gecko: C++ and Rust. C++ is similar to C with more features for object-oriented programming, like classes or interfaces. It suffers from the same memory safety issues as C, such as buffer overflows due to using the same memory paradigms.

Rust is a low-level programming language developed by Mozilla in 2010 [44], although it has since developed its own independent community. Its main goal is to address the shortcomings of C++ in terms of memory safety [45]. It is mainly an imperative programming language and provides features familiar to C developers such as structs or enums. However, it also provides object-oriented programming features such as traits, which are similar to interfaces. Functional programming features such as pattern matching or lambdas can also be found.

The Firefox code makes use of `cbindgen` to make Rust code interact with C++ in the source code. It does this by generating C++ bindings that can be used directly from C++ code.

3.2 Anatomy of a browser engine

In order to describe the implementation, it is useful to first provide an in-breadth overview of how a browser rendering engine renders web pages. This happens in 5 main steps, which are roughly similar for all 3 major rendering engines. Most of the contents in this section are summarised from Lin Clark's Code Cartoons blog post [46].

The first step is parsing, HTML and CSS files are parsed into data structures that are understood by the rendering engine. The structures representing HTML are referred to as the DOM (Document Object Model). For this project, only CSS parsing is relevant and it is done by the style system.

The second step is styling. The CSS engine figures out which CSS properties should apply to each element on the web page, for which the process is not relevant to the project. Also, in each CSS property, since the initial value specified by the file may contain variables, different units or calculation functions, that value needs to be normalised into a final value, called the computed value. An example would be `calc(45deg - 1turn)` being computed to `-315deg`. The way normalisation should be done is not precisely specified by the CSS specification, meaning that different browser engines can differ to this regard by doing what is convenient for them. The only guideline given by the specification is to try to normalise to a value as short as possible.

The third step is layout. The rendering engine computes the on-screen size and position of each box on the webpage. Boxes can be elements in the HTML document, but also parts of those elements such as lines of text or list markers. This part is not very relevant to the project, since conic gradients don't affect the size or position of boxes on the screen.

The fourth step is painting. Each box is painted using the styling and layout information computed at previous steps. The painting is potentially done on different layers, making it possible to repaint a single

layer independently.

The last step is rendering, which is essentially taking the layers from the previous step and rendering them as one single image. Some CSS properties only are applied at this step, such as `transform` (which performs translations, rotations and some other transforms), or `animation`, since it is more convenient and performant for the browser engine to do so.

In Gecko, Firefox's rendering engine, the rendering step is fairly complicated, because Gecko has multiple graphics backends, which will be detailed in their respective implementation section. APIs are used to abstract away different backends. Although in the future, the plan is to replace all existing backends with WebRender. Here is a rough diagram representing the rendering flow:

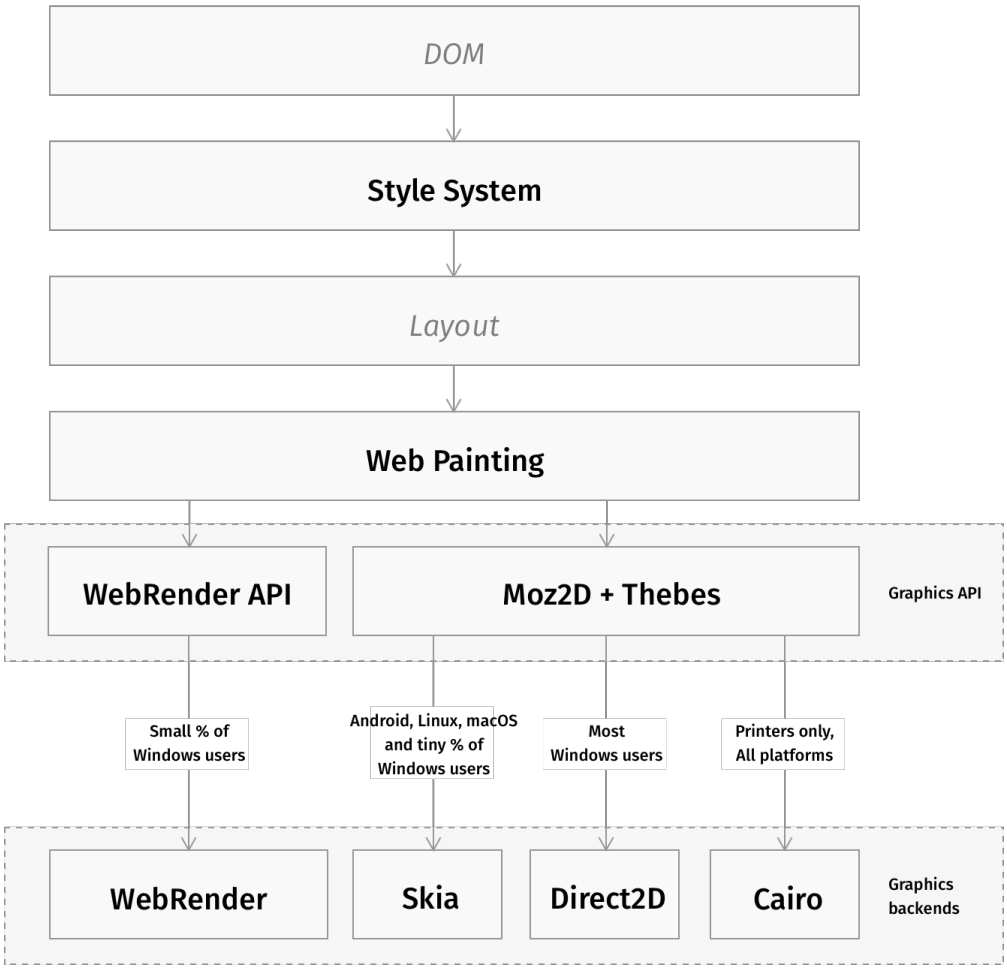


Figure 3.1: How rendering works in Gecko

The grayed out boxes are steps untouched by this project.

3.3 Shaders

For the WebRender implementation, it is useful to describe what a graphics shader is. When doing computer graphics, shaders are programs that run at different points of the rendering pipeline, usually on the GPU [47]. They are often used in computer games or in cinema post-processing. WebRender uses the "OpenGL Shading Language" for its shaders, also abbreviated as as GLSL [48].

There are many different types of shaders but this section will only cover types relevant to this project: fragment shaders and vertex shaders.

Fragments are regions of the 3D scene that can render as one pixel on screen. The fragment shader runs for each fragment of the scene, with the properties of the fragment as input and the color of the pixel as output. [49]

The vertex shader, on the other hand, is ran once for each of the vertices. It is used to compute the position and data of anything related to the vertex [50]. Since it is ran less often than the fragment shader, it is usually good practice to pre-compute vertex data in there when possible.

The following diagram, taken from the open.gl website [51], is a good illustration of the graphics pipeline:

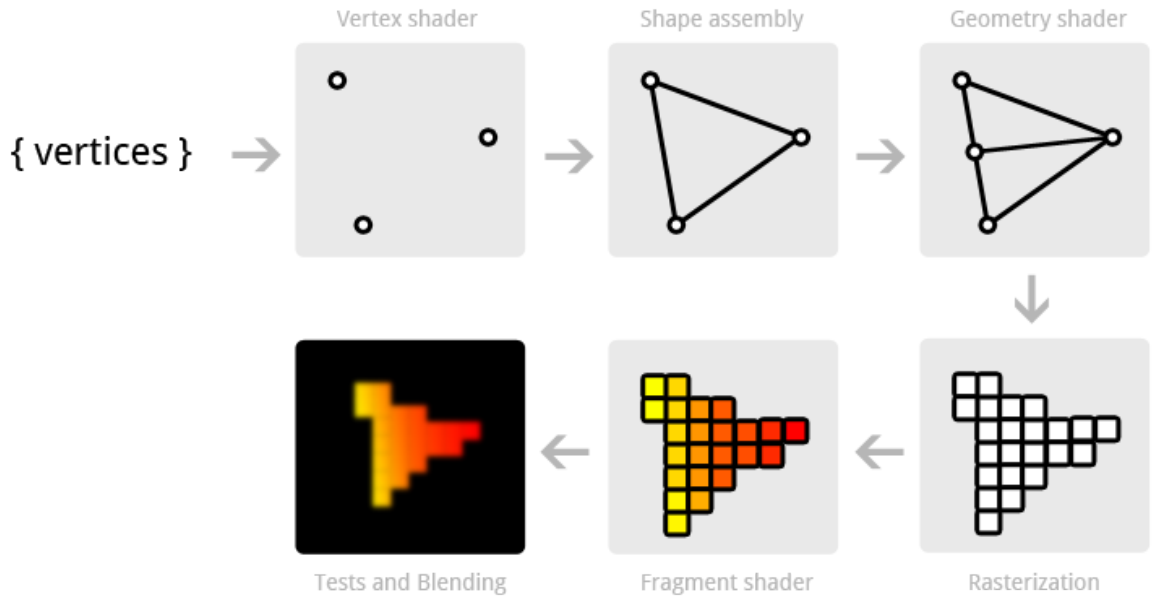


Figure 3.2: Graphics pipeline

Chapter 4

Implementation

This section will describe the implementation done in Gecko, ordered by topic. For historical purposes, the list of merged commits related to this project can be found in appendix C.

4.1 Style System

The CSS engine in Gecko taking care of the styling step is mostly written in Rust. Most of it is shared with the Servo browser engine, an experimental browser engine by Mozilla, also written in Rust [52]. The code is located in the `servo/components/style/` directory.

The part handling CSS values, located in the `servo/components/style/values` directory, is split in multiple directories, for which three are directly relevant to this project:

- `generics`: This directory defines the generic interfaces used by the other directories. Class names in this directory may be prefixed with `Generic`.
- `specified`: This directory contains the implementation for specified values (values specified by the developer) as defined in the specification [53] and has the code for parsing those values.
- `computed`: This directory contains the implementation for computed values, which are the specified values with different types normalised and calculations resolved [54].

There are also two directories not directly relevant to this project:

- `resolved`: The resolved CSS value is the value returned by the `getComputedStyle()` function. Most of the time, it is the computed value, but there are some special cases for web compatibility [55]. For the `conic-gradient` case, while colors have a different resolved value, that bit was already implemented for other color-related properties, so no more work on resolved values is needed.
- `animated`: This directory defines types for animations and transitions. Image values like `conic-gradient` cannot be animated or transitioned [56], so no work is needed for this directory.

There is some pre-existing code handling gradients, for linear and radial gradients, that was re-usable. However adding in conic gradients to the style system was not a simple task due to numerous differences that conic gradients have.

4.1.1 Adding the AngleOrPercentage type

The first of these differences is the need to support angles or percentages for color stop positions. Since no other CSS feature currently uses `<angle-percentage>`, the `AngleOrPercentage` type needed to be added to the style system. The complete diff can be viewed on Phabricator [57].

Naming

The name `AnglePercentage` was ruled out even though `<length-percentage>` is named `LengthPercentage` in the source code, since `AngleOrPercentage` does not support mixing angles and percentages in the `calc`

CSS function, while `LengthPercentage` does support mixing lengths and percentages. An example of mixing angles and percentages is:

```
conic-gradient(red calc(10deg + 25%), gold)
```

While the specification says that this should be supported [58], from manual testing, no other web browser currently supports it (see appendix D). Also, the use cases for mixing angles and percentages is low, since the above angle could be expressed as `calc(10deg + 90deg)` or directly as `100deg`. It is also not trivial to support, so this was left out for the initial implementation.

Definition

The implementation for `AngleOrPercentage`, both for the computed and specified versions, consists of a Rust enum saying it is either a `Percentage` or an `Angle`, and includes methods on top that re-use code from the two sub-types.

```
pub enum AngleOrPercentage {
    Percentage(Percentage),
    Angle(Angle),
}
```

Specified value implementation

The specified value implementation contains the parsing code. The `parse_internal` function does the following:

1. Try to parse the value as a percentage
2. Return that percentage value if successful
3. Try to parse the value as an angle value

There is a `parse_with_unitless` and a `parse` function which both call the `parse_internal` function with a different "allow unitless zero" parameter. The distinction is subtle, but the parameter determines whether to allow `0` without any unit. Even though zeroes without units are convenient, they are ambiguous just like any unitless number, especially for properties that accept multiple numeric values as full value. As an example, the relevant `csswg-drafts` Github issue [59] uses the `offset` CSS property value, defined as:

```
[ offset-position? [ offset-path [ <length-percentage> || offset-rotate ]? ]? ]!
[ / offset-anchor ]?
```

where `offset-rotate` is `[ auto | reverse ] || <angle>`.

A value like `offset: ray(180deg) 0;` (where `ray(180deg)` is `offset-path`) then becomes ambiguous since `0` could be either the `<length-percentage>` or the `offset-rotate` value. It was decided in that Github issue that unitless zeroes should only be allowed for CSS transforms, filters, gradients for compatibility reasons.

As for conic gradients, the specification explicitly says unitless zeroes should be allowed for color stop positions [60], presumably for consistency with other gradients. However, the default parsing method for `AngleOrPercentage` remains without unitless zero support to avoid accidentally supporting it for future specifications using this type.

The full code for the specified value implementation can be found in appendix E.1.

Computed value implementation

The computed value implementation has two methods from the `ToComputedValue` trait:

- `to_computed_value`, which resolves a specified value to a computed value.
- `from_computed_value`, which reconstructs a specified value from a computed value.

Both of these methods pattern match on the value then call pre-existing methods from the sub-types:

```
impl ToComputedValue for specified::AngleOrPercentage {
    type ComputedValue = AngleOrPercentage;

    #[inline]
    fn to_computed_value(&self, context: &Context) -> AngleOrPercentage {
        match *self {
            specified::AngleOrPercentage::Percentage(percentage) => {
                AngleOrPercentage::Percentage(percentage.to_computed_value(context))
            },
            specified::AngleOrPercentage::Angle(angle) => {
                AngleOrPercentage::Angle(angle.to_computed_value(context))
            },
        }
    }
    #[inline]
    fn from_computed_value(computed: &AngleOrPercentage) -> Self {
        match *computed {
            AngleOrPercentage::Percentage(percentage) => {
                specified::AngleOrPercentage::Percentage(ToComputedValue::from_computed_value(
                    &percentage,
                ))
            },
            AngleOrPercentage::Angle(angle) => {
                specified::AngleOrPercentage::Angle(ToComputedValue::from_computed_value(&angle))
            },
        }
    }
}
```

4.1.2 Refactoring gradient code

Once the `AngleOrPercentage` type was added, the existing gradient implementation needed to be more generic to allow using this type instead of `LengthPercentage`. To do this, instances of `LengthPercentage` were first replaced with Rust generic types [61], in the `ColorStop` and `GradientItem` implementations. These pre-existing classes are respectively analogous to the `<color-stop>` type and the type of `<color-stop-list>` items, both from the CSS Images specification [62].

To illustrate the change, here is an extract from the full diff [63]:

```
--- a/servo/components/style/values/generics/image.rs
+++ b/servo/components/style/values/generics/image.rs
@@ -175,48 +175,48 @@ pub enum ShapeExtent {
-pub enum GenericGradientItem<Color, LengthPercentage> {
+pub enum GenericGradientItem<Color, T> {
     /// A simple color stop, without position.
     SimpleColorStop(Color),
     /// A complex color stop, with a position.
     ComplexColorStop {
         /// The color for the stop.
         color: Color,
         /// The position for the stop.
-        position: LengthPercentage,
+        position: T,
     },
     /// An interpolation hint.
-    InterpolationHint(LengthPercentage),
+    InterpolationHint(T),
 }
```

```
 pub use self::GenericGradientItem as GradientItem;

 /// A color stop.
 /// <https://drafts.csswg.org/css-images/#typedef-color-stop-list>
 #[derive(
     Clone, Copy, Debug, MallocSizeOf, PartialEq, ToComputedValue, ToCss, ToResolvedValue, ToShmem,
 )]
-pub struct ColorStop<Color, LengthPercentage> {
+pub struct ColorStop<Color, T> {
     /// The color of this stop.
     pub color: Color,
     /// The position of this stop.
-    pub position: Option<LengthPercentage>,
+    pub position: Option<T>,
 }

-impl<Color, LengthPercentage> ColorStop<Color, LengthPercentage> {
+impl<Color, T> ColorStop<Color, T> {
     /// Convert the color stop into an appropriate `GradientItem`.
     #[inline]
-    pub fn into_item(self) -> GradientItem<Color, LengthPercentage> {
+    pub fn into_item(self) -> GradientItem<Color, T> {
         match self.position {
```

This change was not sufficient, since the gradient type definition was split into two parts: `Gradient` and `GradientKind`. `Gradient` contains information shared between all types of gradients, while `GradientKind` contains information specific to each gradient type (linear/radial), as shown below:

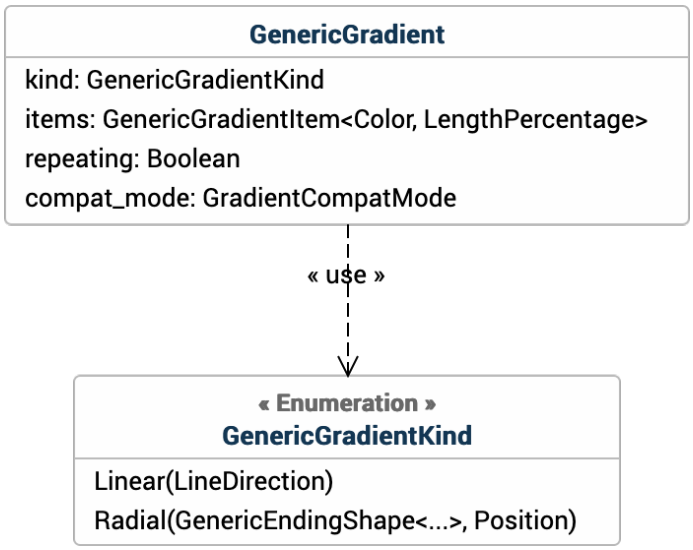


Figure 4.1: Class diagram before refactor

Given that the `items` field would differ for conic gradients, due to using a different unit type, it would need to move to `GradientKind`, but that would leave only `repeating` and `compat_mode` in the shared `Gradient` structure.

As mentioned in chapter 1, there are prefixes like `-moz-linear-gradient` or `-webkit-gradient` that need to remain supported, for which the parsing of the function can differ. These different parsing modes are represented via the compatibility mode. However, prefixes are not relevant to conic gradients, leaving

only the `repeating` field shared between gradients. Hence, the `GradientKind` was merged back into `Gradient`. The new structure now looks like this:

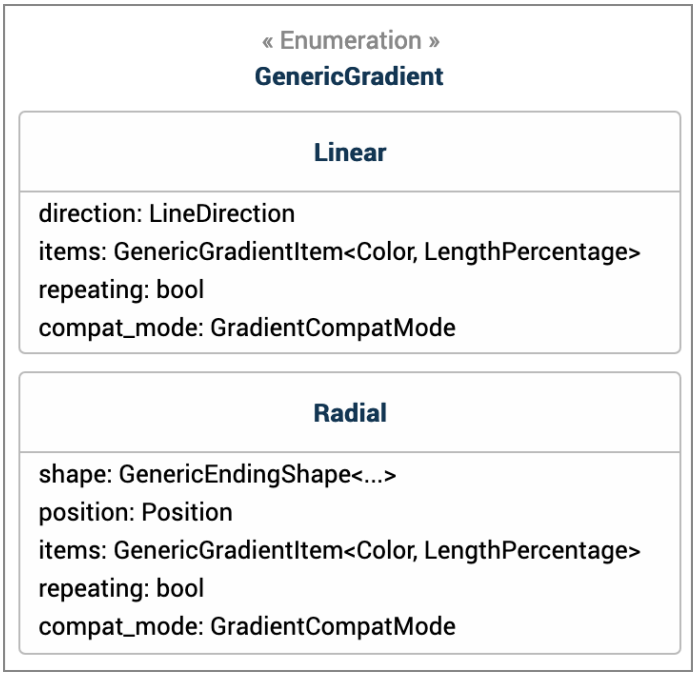


Figure 4.2: Class diagram after refactor

Changing the structure required adapting most of the existing code in the style system. The web painting code (in C++), which uses those structures via `cbindgen`, also needed to be adapted. For instance, `aGradient.kind.AsRadial()` was changed to `aGradient.AsRadial()`.

The full definition of the structures can be found in appendix F and the complete diff of the refactor can be viewed on Phabricator [64].

4.1.3 Parsing

Now that the gradient implementation is generic enough, the conic gradient parsing can now be implemented. First, the `Gradient` structure needs to be augmented to know what a conic gradient is. The conic gradient type is defined below:

```
/// A conic gradient.
Conic {
    /// Start angle of gradient
    angle: Angle,
    /// Center of gradient
    position: Position,
    /// The color stops and interpolation hints.
    items: crate::OwnedSlice<GenericGradientItem<Color, AngleOrPercentage>>,
    /// True if this is a repeating gradient.
    repeating: bool,
},
```

To support the `conic-gradient` and `repeating-conic-gradient` functions, cases need to be added to detect them, in the gradient `parse` function:

```
let (shape, repeating, compat_mode) = match_ignore_ascii_case! { &func,
    // [...]
```

```
+    "conic-gradient" if static_prefs::pref!("layout.css.conic-gradient.enabled") => {
+        (Shape::Conic, false, GradientCompatMode::Modern)
+    },
+    "repeating-conic-gradient" if static_prefs::pref!("layout.css.conic-gradient.enabled") => {
+        (Shape::Conic, true, GradientCompatMode::Modern)
+    },
     // [...]
};
 Ok(input.parse_nested_block(|i| {
     Ok(match shape {
         // [...]
+        Shape::Conic => Self::parse_conic(context, i, repeating)?,
     })
 })?)
```

`static_prefs::pref!("layout.css.conic-gradient.enabled")` checks that the feature flag is turned on. If it is turned off, conic gradients will be ignored.

Once the cases are handled, the `conic-gradient` parsing needs to be defined. The steps to parse the `conic-gradient` function are:

1. Try to parse a `from` token followed by an angle, that is potentially an unitless zero.
2. Try to parse an `at` token followed by a position.
3. If an angle or a position was found, expect a comma.
4. If not specified, use zero as the default angle, and center as default position
5. Parse the color stops with angles or percentages as positions, and potentially unitless zeros
6. Reject the function if there are less than 2 color stops (a gradient with 1 stop is pointless since it is just a solid color)
7. Return a conic gradient structure

The code for the parsing function can be seen below:

```
fn parse_conic<'i, 't>(
    context: &ParserContext,
    input: &mut Parser<'i, 't>,
    repeating: bool,
) -> Result<Self, ParseError<'i>> {
    let angle = input.try(|i| {
        i.expect_ident_matching("from")?;
        // Spec allows unitless zero start angles
        // https://drafts.csswg.org/css-images-4/#valdef-conic-gradient-angle
        Angle::parse_with_unitless(context, i)
    });
    let position = input.try(|i| {
        i.expect_ident_matching("at")?;
        Position::parse(context, i)
    });
    if angle.is_ok() || position.is_ok() {
        input.expect_comma()?;
    }

    let angle = angle.unwrap_or(Angle::zero());
    let position = position.unwrap_or(Position::center());
    let items = generic::GradientItem::parse_comma_separated(context, input,
↪   AngleOrPercentage::parse_with_unitless)?;

    if items.len() < 2 {
        return Err(input.new_custom_error(StyleParseErrorKind::UnspecifiedError));
```

```
    }

    Ok(Gradient::Conic {
        angle,
        position,
        items,
        repeating,
    })
}
```

4.1.4 Serialisation

Now that the gradient is parsed into a structure that is understood by the style system, the style system needs to be able to serialise the structure back to CSS if asked by the following JavaScript methods: `getComputedStyle(element)` or `element.style`. To do this, the `ToCss` Rust trait must be implemented for conic gradients. The method takes a gradient structure and transforms it back to a CSS string.

```
 fn to_css<W>(&self, dest: &mut CssWriter<W>) -> fmt::Result
 where
     W: Write,
 {
     // [...]
     match *self {
         // [...]
+        Gradient::Conic { ref angle, ref position, ref items, .. } => {
+            dest.write_str("conic-gradient(")?;
+            let omit_angle = angle.is_zero();
+            let omit_position = position.is_center();
+            if !omit_angle {
+                dest.write_str("from ")?;
+                angle.to_css(dest)?;
+                if !omit_position {
+                    dest.write_str(" ")?;
+                }
+            }
+            if !omit_position {
+                dest.write_str("at ")?;
+                position.to_css(dest)?;
+            }
+            let mut skip_comma = omit_angle && omit_position;
+            for item in &**items {
+                if !skip_comma {
+                    dest.write_str(", ")?;
+                }
+                skip_comma = false;
+                item.to_css(dest)?;
+            }
+        },
+    }
     dest.write_str(")")
 }
```

Since the specification says CSS structures should be serialised to the most simple form when possible, the angle and the position are omitted if they use their default values [65]. Although it was not a big issue, it is worth noting that the position was not omitted for radial gradients when it was using the default value (the center of the element), Chrome and Safari also do not respect this [66]. This was fixed in Firefox along with the conic gradient implementation [67]. For instance, `radial-gradient(at 50% 50%, red, green)` now gets simplified to `radial-gradient(red, green)`.

To fix this for both conic and radial gradients, a new Rust trait named `PositionComponent` was de-

fined in the `generics` directory. The trait contains a `is_center` method, checking whether the position component (horizontal or vertical) is the central one:

```
/// Implements a method that checks if the position is centered.
pub trait PositionComponent {
    /// Returns if the position component is 50% or center.
    /// For pixel lengths, it always returns false.
    fn is_center(&self) -> bool;
}
```

The following classes implement this trait:

- `Position` in the `generics` directory, where the implementation checks whether both the horizontal and vertical components of the position are central.
- `LengthPercentage` in the `computed` directory, where the implementation converts the value to a percentage, then pattern matches against it.
- `PositionComponent` in the `specified` directory, for the `<position>` CSS type [68], where the implementation pattern matches against the type of value used.

For instance, here is the implementation for the `LengthPercentage` class:

```
impl GenericPositionComponent for LengthPercentage {
    fn is_center(&self) -> bool {
        match self.to_percentage() {
            Some(Percentage(per)) => per == 0.5,
            _ => false,
        }
    }
}
```

Now that this `PositionComponent` trait is implemented, using it in the conic gradient code consists of omitting the position if central, when serialising:

```
            Gradient::Conic { ref angle, ref position, ref items, .. } => {
                dest.write_str("conic-gradient(")?;
-               if !angle.is_zero() {
+               let omit_angle = angle.is_zero();
+               let omit_position = position.is_center();
+               if !omit_angle {
                    dest.write_str("from ")?;
                    angle.to_css(dest)?;
-                   dest.write_str(" ")?;
+                   if !omit_position {
+                       dest.write_str(" ")?;
+                   }
                }
-               dest.write_str("at ")?;
-               position.to_css(dest)?;
+               if !omit_position {
+                   dest.write_str("at ")?;
+                   position.to_css(dest)?;
+               }
+               let mut skip_comma = omit_angle && omit_position;
                for item in &**items {
-                   dest.write_str(", ")?;
+                   if !skip_comma {
+                       dest.write_str(", ")?;
+                   }
+                   skip_comma = false;
                    item.to_css(dest)?;
```

```
            }
        },
```

As the code above shows, it is important to take extra care to avoid including the associated keyword (`from` or `at`) and whitespace if omitting the angle or position. It is also important to skip the first comma if both the angle and position are omitted, to prevent ending up with an invalid CSS value like `conic-gradient(, red, blue)`.

The full diff can be found on Phabricator for completeness [67].

4.1.5 Testing

Existing parsing and serialisation tests were augmented with testcases for correct syntax, but also for invalid syntax:

```
/* Invalid units */
"conic-gradient(red, blue 50px, yellow 30px)",
"conic-gradient(from 0%, black, white)",
"conic-gradient(from 60%, black, white)",
"conic-gradient(from 40px, black, white)",
"conic-gradient(from 50, black, white)",
"conic-gradient(at 50deg, black, white)",
"conic-gradient(from 40deg at 50deg, black, white)",
"conic-gradient(from 40deg at 50deg 60deg, black, white)",
/* Invalid keywords (or ordering) */
"conic-gradient(at 40% from 50deg, black, white)",
"conic-gradient(to 50deg, black, white)",
/* Conic gradients should not support prefixed syntax */
"-webkit-gradient(conic, 1 2, 3 4, color-stop(0, lime))",
"-webkit-conic-gradient(red, blue)",
"-moz-conic-gradient(red, blue)",
"-webkit-repeating-conic-gradient(red, blue)",
"-moz-repeating-conic-gradient(red, blue)",
```

4.2 Web Painting

Now that the gradient is parsed to a structure understood by the code, the painting phase needs to happen.

The web painting code is all in C++ and is in the `layout/painting/` directory. The relevant code for gradients is located in the `nsCSSGradientRenderer` class, and is used in `nsImageRenderer` class for all image related CSS properties (such as `background-image` or `mask-image`), except for `border-image` which `nsCSSBorderImageRenderer` handles.

This code knows about the size and position of the element on the screen, which was not available in the style system. This allows some information like the position of the conic gradient to be resolved, which is needed for painting.

After the refactor mentioned in section 4.1.2, the `nsCSSRenderingGradients.cpp` file (containing `nsCSSGradientRenderer`) was changed over multiple commits (listed in chronological order), for which the combined diff can be found in appendix G:

1. Tim Nguyen - Bug 1615862 - Handle conic-gradients in nsCSSGradientRenderer for WebRender. r=emilio,mstange (`https://phabricator.services.mozilla.com/D63018`)
2. Tim Nguyen - Bug 1616587 - Implement conic-gradient for Skia graphics backend. r=lsalzman (`https://phabricator.services.mozilla.com/D63415`)
3. Tim Nguyen - Bug 1620328 - Set conic-gradient angle range on Skia. r=lsalzman (`https://phabricator.services.mozilla.com/D65910`)

The first step was setting up the ground work for supporting conic gradients in both `nsCSSGradientRenderer` and `nsCSSBorderImageRenderer`:

- Fields, variables and methods for properties specific to conic gradients (center and start angle) were added to the class.
- Extra cases were added where relevant.

The diff below shows an example of the above changes done on the `nsCSSGradientRenderer::Create` method:

```
 nsCSSGradientRenderer nsCSSGradientRenderer::Create(
     nsPresContext* aPresContext, ComputedStyle* aComputedStyle,
     const StyleGradient& aGradient, const nsSize& aIntrinsicSize) {
   auto srcSize = CSSSize::FromAppUnits(aIntrinsicSize);

   // Compute "gradient line" start and end relative to the intrinsic size of
   // the gradient.
-  CSSPoint lineStart, lineEnd;
-  CSSCoord radiusX = 0, radiusY = 0;  // for radial gradients only
+  CSSPoint lineStart, lineEnd, center;  // center is for conic gradients only
+  CSSCoord radiusX = 0, radiusY = 0;    // for radial gradients only
+  float angle = 0.0;                    // for conic gradients only
   if (aGradient.IsLinear()) {
     Tie(lineStart, lineEnd) =
         ComputeLinearGradientLine(aPresContext, aGradient, srcSize);
-  } else {
+  } else if (aGradient.IsRadial()) {
     Tie(lineStart, lineEnd, radiusX, radiusY) =
         ComputeRadialGradientLine(aGradient, srcSize);
+  } else {
+    MOZ_ASSERT(aGradient.IsConic());
+    Tie(center, angle) = ComputeConicGradientProperties(aGradient, srcSize);
   }
   // Avoid sending Infs or Nans to downwind draw targets.
   if (!lineStart.IsFinite() || !lineEnd.IsFinite()) {
     lineStart = lineEnd = CSSPoint(0, 0);
   }
+  if (!center.IsFinite()) {
+    center = CSSPoint(0, 0);
+  }
   CSSCoord lineLength =
       NS_hypot(lineEnd.x - lineStart.x, lineEnd.y - lineStart.y);

   // Build color stop array and compute stop positions
   nsTArray<ColorStop> stops =
       ComputeColorStops(aComputedStyle, aGradient, lineLength);

   ResolveMidpoints(stops);
@@ -672,16 +723,21 @@ nsCSSGradientRenderer nsCSSGradientRende
       aPresContext->CSSPixelsToDevPixels(lineStart.y),
   };
   renderer.mLineEnd = {
       aPresContext->CSSPixelsToDevPixels(lineEnd.x),
       aPresContext->CSSPixelsToDevPixels(lineEnd.y),
   };
   renderer.mRadiusX = aPresContext->CSSPixelsToDevPixels(radiusX);
   renderer.mRadiusY = aPresContext->CSSPixelsToDevPixels(radiusY);
+  renderer.mCenter = {
+      aPresContext->CSSPixelsToDevPixels(center.x),
+      aPresContext->CSSPixelsToDevPixels(center.y),
+  };
+  renderer.mAngle = angle;
```

```
   return renderer;
 }
```

Less importantly, some methods have been changed to C++ template methods, to support both angular and length color stops. This was not needed during the refactor from section 4.1.2, since none of the web painting code was calling those methods with angular color stops yet.

The next step was adding hooks to the WebRender API in pre-existing methods. Parameters are initialised in `BuildWebRenderParameters`, while the call to the API is done in `BuildWebRenderDisplayItems`.

In the second and third commits, since the Moz2D graphics pattern was added (this will be described in its respective section), it was possible to add handling for the conic gradient case to the `Paint` method of `nsCSSGradientRenderer`:

```
--- a/layout/painting/nsCSSRenderingGradients.cpp
+++ b/layout/painting/nsCSSRenderingGradients.cpp
@@ -946,28 +946,27 @@ void nsCSSGradientRenderer::Paint(gfxCon
   // Create the gradient pattern.
   RefPtr<gfxPattern> gradientPattern;
   gfxPoint gradientStart;
   gfxPoint gradientEnd;
   if (mGradient->IsLinear()) {
     // [...]
   } else if (mGradient->IsRadial()) {
     // [...]
   } else {
-    return;
+    gradientPattern = new gfxPattern(mCenter.x, mCenter.y, mAngle, stopOrigin, stopEnd);
   }
   // Use a pattern transform to take account of source and dest rects
   matrix.PreTranslate(gfxPoint(mPresContext->CSSPixelsToDevPixels(aSrc.x),
                                mPresContext->CSSPixelsToDevPixels(aSrc.y)));
   matrix.PreScale(
       gfxFloat(nsPresContext::CSSPixelsToAppUnits(aSrc.width)) / aDest.width,
       gfxFloat(nsPresContext::CSSPixelsToAppUnits(aSrc.height)) / aDest.height);
   gradientPattern->SetMatrix(matrix);
```

Equivalent changes were also done in `nsCSSBorderImageRenderer`.

4.3 WebRender

WebRender is one of Firefox graphics backends developed by Mozilla written in Rust, its aim is to make full use of the GPU to render the web. It is used by Firefox and Servo, but can also be used independently.

It is enabled by default on a subset of Windows 10 devices [69] and can be enabled using the `gfx.webrender.enabled` flag on other platforms as well [70]. In the future, the Firefox graphics team plans to use WebRender everywhere and replace all other backends as much as possible [69].

Most of the implementation for the WebRender involved re-using boilerplate code from other gradient types, adapted for the conic gradient parameters, so some of the following sub-sections may not be very detailed. The main work requiring investigation was the maths for the fragment shader.

The full diff may be found at [71] for completeness.

4.3.1 Boilerplate

These files were changed to set up the boilerplate code, which can be categorised in the following categories:

- Primitive types, which are atomic types that can be drawn or cached on the hardware [72]; along with their related code.

  gfx/wr/webrender/src/batch.rs (83 lines added/changed)
  gfx/wr/webrender/src/picture.rs (3 lines)
  gfx/wr/webrender/src/prim_store/gradient.rs (220 lines)
  gfx/wr/webrender/src/prim_store/interned.rs (2 lines)
  gfx/wr/webrender/src/prim_store/mod.rs (89 lines)
  gfx/wr/webrender/src/render_backend.rs (4 lines)

- Brush types associated to the OpenGL shader

  gfx/wr/webrender/res/brush.glsl (2 lines)
  gfx/wr/webrender/res/brush_multi.glsl (32 lines)
  gfx/wr/webrender/src/gpu_types.rs (9 lines)
  gfx/wr/webrender/src/renderer.rs (7 lines)
  gfx/wr/webrender/src/shade.rs (20 lines)

4.3.2 WebRender API

The aim of the WebRender API is to expose internal methods that are used to push new display items to the display list. The display list is a list of items to be displayed on screen, where each item is combined up from primitives [73].

API bindings

The API bindings are Rust methods that the C++ WebRender API uses to interact with WebRender. In this case, the relevant bindings are:

- `wr_dp_push_conic_gradient`
- `wr_dp_push_border_conic_gradient`

As their name suggests, those methods push new conic gradient items (image and border-image) to the display list. They are located in `gfx/webrender_bindings/src/bindings.rs` and are exposed to C++ via the following prefix on the method signatures: `pub extern "C"`.

For those bindings to work, relevant display list item types for conic gradient background images and border images were added in the following files, based on pre-existing code:

gfx/wr/webrender/src/scene_building.rs (84 lines)
gfx/wr/webrender_api/src/api.rs (1 line)
gfx/wr/webrender_api/src/display_item.rs (25 lines)
gfx/wr/webrender_api/src/display_list.rs (39 lines)
gfx/wr/webrender_api/src/gradient_builder.rs (21 lines)

C++ definition

Once the underlying Rust code was done, the corresponding C++ methods were added and called from the web painting code:

```
void DisplayListBuilder::PushConicGradient(
    const wr::LayoutRect& aBounds, const wr::LayoutRect& aClip,
    bool aIsBackfaceVisible, const wr::LayoutPoint& aCenter, const float aAngle,
    const nsTArray<wr::GradientStop>& aStops, wr::ExtendMode aExtendMode,
    const wr::LayoutSize aTileSize, const wr::LayoutSize aTileSpacing) {
  wr_dp_push_conic_gradient(mWrState, aBounds, MergeClipLeaf(aClip),
                            aIsBackfaceVisible, &mCurrentSpaceAndClipChain,
                            aCenter, aAngle, aStops.Elements(), aStops.Length(),
                            aExtendMode, aTileSize, aTileSpacing);
}

void DisplayListBuilder::PushBorderConicGradient(
```

```
    const wr::LayoutRect& aBounds, const wr::LayoutRect& aClip,
    bool aIsBackfaceVisible, const wr::LayoutSideOffsets& aWidths, bool aFill,
    const wr::LayoutPoint& aCenter, const float aAngle,
    const nsTArray<wr::GradientStop>& aStops, wr::ExtendMode aExtendMode,
    const wr::LayoutSideOffsets& aOutset) {
  wr_dp_push_border_conic_gradient(
      mWrState, aBounds, MergeClipLeaf(aClip), aIsBackfaceVisible,
      &mCurrentSpaceAndClipChain, aWidths, aFill, aCenter, aAngle,
      aStops.Elements(), aStops.Length(), aExtendMode, aOutset);
}
```

The parameters used are the same as for other gradient types, but adapted for conic gradients.

4.3.3 Shaders

The boilerplate code for shaders was the same as the one for radial gradients, but with the mathematics and parameters adapted for conic gradients.

The file containing the shaders is located at `gfx/wr/webrender/res/brush_conic_gradient.glsl` [74].

Fetching the parameters

The code fetching the conic gradient parameters to store them in a `struct` in memory can be found below:

```
struct ConicGradient {
    vec2 center_point;
    vec2 start_end_offset;
    float angle;
    int extend_mode;
    vec2 stretch_size;
};

ConicGradient fetch_gradient(int address) {
    vec4 data[2] = fetch_from_gpu_cache_2(address);
    return ConicGradient(
        data[0].xy,
        data[0].zw,
        float(data[1].x),
        int(data[1].y),
        data[1].zw
    );
}
```

The `vec4` and `vec2` are structures that respectively store 4 and 2 numbers. They are used to their fullest to prevent wasting unnecessary memory using a process called "swizzling". For instance, in a `vec4` structure, `x`, `y`, `z`, or `w`, refer to the first, second, third, and fourth components, respectively. Letters can also be combined: `xy` will refer to a `vec2` with both the first and second elements in that order [75].

Vertex shader

For reference, the signature of the conic gradient vertex shader is as follows:

```
void conic_gradient_brush_vs(
    VertexInfo vi,
    int prim_address,
    RectWithSize local_rect,
    RectWithSize segment_rect,
    ivec4 prim_user_data,
    int specific_resource_address,
    mat4 transform,
    PictureTask pic_task,
```

```
    int brush_flags,
    vec4 texel_rect
)
```

The vertex shader first does its main job by computing the position of the vertex, using code that is identical to other gradients:

```
if ((brush_flags & BRUSH_FLAG_SEGMENT_RELATIVE) != 0) {
    V_POS = (vi.local_pos - segment_rect.p0) / segment_rect.size;
    V_POS = V_POS * (texel_rect.zw - texel_rect.xy) + texel_rect.xy;
    V_POS = V_POS * local_rect.size;
} else {
    V_POS = vi.local_pos - local_rect.p0;
}
```

The gradient parameters are then fetched from the memory for future use in the fragment shader. The parameters specific to conic gradients are the following:

- center: the center of the gradient
- angle: the angle of the gradient in radians
- start offset: the position of the first color stop
- end offset: the position of the last color stop

The code doing this step is as follows:

```
ConicGradient gradient = fetch_gradient(prim_address);

V_CENTER = gradient.center_point;
V_ANGLE = gradient.angle;
V_START_OFFSET = gradient.start_end_offset.x;
V_END_OFFSET = gradient.start_end_offset.y;
```

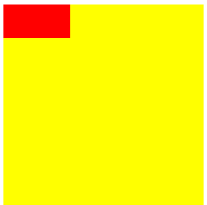
Other parameters are also stored, but they are not directly relevant to this project since they are not specific to conic gradients.

Fragment shader

The difference in the fragment shader compared to the other gradients were the mathematics used to map each fragment to a color. Multiple versions were investigated before reaching the final version. For simplicity, the example that will be used across this section is `conic-gradient(red, yellow)`.

There was already a function getting the gradient color based on a color stop offset called `sample_gradient`, so for the current example, inputting `0.0` would output red, while `1.0` would output yellow. With that function, the remaining goal was to figure out how to map x/y coordinates of a pixel to an offset.

The first step was figuring out the coordinate system, which was checked with this experiment:



```
float offset;
if (pos.x < 100 && pos.y < 50) {
  offset = 0.0;
} else {
  offset = 1.0;
}
```

Figure 4.3: Axis experiment

It can be deduced that the coordinate system is similar to many other ones used in software:

- $(0, 0)$ is the top left corner

- x axis grows to the right
- y axis grows to the bottom

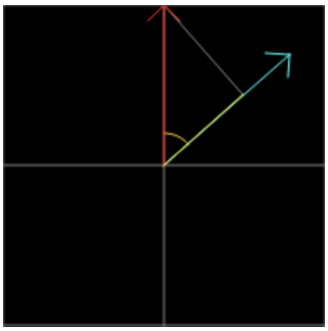


Figure 4.4: Dot product visualisation from `https://falstad.com/dotproduct`

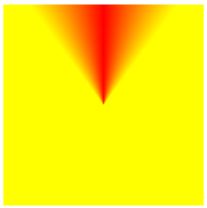
Most following experiments were based the dot product between the start angle vector (**a**, visualised in red) and the current angle vector (**b**, visualised in blue). As a quick reminder, the dot product between two vectors **a** and **b** is defined by:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \ \|\mathbf{b}\| \cos\theta$$

With some shuffling, the current angle (θ, visualised in orange) can be found:

$$\theta = \arccos\left((\mathbf{a} \cdot \mathbf{b}) \div (\|\mathbf{a}\| \ \|\mathbf{b}\|)\right)$$

To provide an offset from 0 to 1, the current angle must be divided by 2π. This was the initial attempt of translating the formula to code, it assumes the start angle parameter to be 0 for simplicity:

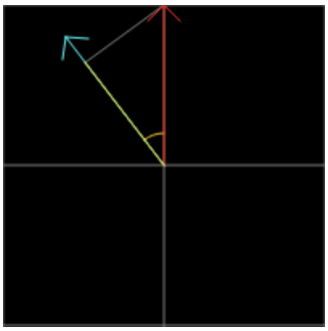


```
vec2 start_angle_vector = vec2(0.0, -V_CENTER.y);
float dot_prod = dot(start_angle_vector, pos - V_CENTER);
float current_angle = mod(
    acos(dot_prod / (V_CENTER.y * length(pos - V_CENTER))),
    2 * PI
);
float offset = current_angle / 2 * PI;
```

Figure 4.5: Initial attempt

This had two main mistakes:

- An overlooked bracketing mistake in the offset division: `current_angle / (2 * PI)` should have been used instead of `current_angle / 2 * PI`.
- When the current offset vector was on the left side, the current angle found from the dot product was the wrong one, the explement of that angle should have been chosen (e.g. the other angle that makes it add up to a full turn). See the figure below:



After fixing both of these problems, the result looked as expected:

```
vec2 current_dir = pos - V_CENTER;
vec2 start_angle_vector = vec2(0.0, -V_CENTER.y);
float dot_prod = dot(start_angle_vector, current_dir);
float current_angle = acos(dot_prod / (V_CENTER.y * length(current_dir)));
if (current_dir.x < 0) {
    current_angle = 2 * PI - current_angle;
}
current_angle = mod(current_angle, 2 * PI);
float offset = current_angle / (2 * PI);
```

Figure 4.6: Result after fixes

The previous code still assumed the start angle parameter to be 0, which needed to be fixed:

```
vec2 current_dir = normalize(pos - V_CENTER);

float start_angle = PI / 2 - V_ANGLE;
vec2 start_vector = normalize(vec2(cos(start_angle), -sin(start_angle)));

float current_angle = acos(dot(start_vector, current_dir));
if (start_vector.x * current_dir.y - current_dir.x * start_vector.y < 0) {
    current_angle = 2 * PI - current_angle;
}
current_angle = mod(current_angle, 2 * PI);

float offset = current_angle / (2 * PI);
```

The formula found was ridiculously complicated, since it was thought out based on Cartesian coordinates. Markus Stange, a Gecko graphics engineer, suggested a simplified formula based on the previous attempt, presumably using polar coordinates:

```
vec2 current_dir = pos - V_CENTER;
float current_angle = atan(current_dir.y, current_dir.x) + (PI / 2 - V_ANGLE);
float offset = mod(current_angle / (2 * PI), 1.0);
```

This was the formula that was initially merged. The history of the formula can be found by browsing through the History tab of the Phabricator revision [71].

With that formula, most cases rendered properly, except for gradients with first and last color stop offsets different than 0% or 100%. An example taken from the polyfill website being:

```
conic-gradient(black 25%, white 0 50%, black 0 75%, white 0);
```

This was fixed in a follow up commit [76] by normalising the offsets based on the first and last offsets, which led to the final version of the fragment shader:

```
Fragment conic_gradient_brush_fs() {
    // [...]
    vec2 current_dir = pos - V_CENTER;
    float current_angle = atan(current_dir.y, current_dir.x) + (PI / 2.0 - V_ANGLE);
    float offset = mod(current_angle / (2.0 * PI), 1.0) - V_START_OFFSET;
    offset = offset / (V_END_OFFSET - V_START_OFFSET);

    vec4 color = sample_gradient(V_GRADIENT_ADDRESS,
                                 offset,
                                 V_GRADIENT_REPEAT);
    // [...]
    return Fragment(color);
}
```

4.3.4 Wrench

Wrench is a debugging tool and testing framework for WebRender: it can replay recordings and execute YAML-based reference tests [77]. Similar to other display item types, two methods needed to be added, along with related boilerplate code:

- `handle_conic_gradient` in `yaml_frame_reader.rs`: handling the YAML `conic-gradient` item, by calling the WebRender API to push a conic gradient item to the display list.
- `conic_gradient_to_yaml` in `yaml_frame_reader.rs`: serialising conic gradient items to YAML, for WebRender recordings.

Here is an example YAML test for conic gradients:

```
---
root:
  items:
    - type: conic-gradient
      bounds: 50 50 300 300
      center: 150 150
      angle: 0.0
      stops: [0.0, red, 1.0, yellow]
```

This is equivalent to `conic-gradient(red, yellow)` on a 300×300 canvas.

The wrench test references are often PNG files, but they can also be another YAML file. References are linked together with the test using a metadata file called `reftest.list`.

Many reference tests were added for WebRender in a separate commit [78]. Since the most relevant tests have been ported to Web Platform Tests, the testcases will be described in that section.

4.4 Moz2D and Thebes

In Firefox, there are many graphics backends, which means it is necessary to glue them together and abstract away the implementation details of each graphics backend. The part doing this is the Moz2D graphics API, which is an cross-platform interface onto the different backends. All current backends are interfaced by Moz2D, except for WebRender which uses its own API for interfacing. The code for Moz2D can be found in the `gfx/2d` directory. Thebes is the API preceding Moz2D, but some parts are still in use today. [79]

Most of the code for Moz2D and Thebes is boilerplate code added along in the Skia implementation commit [80], but the main parts will be mentioned here:

- The pattern class definition in `gfx/2d/2D.h`, describing a conic gradient pattern:

  ```
  class ConicGradientPattern : public Pattern {
   public:
    /// For constructor parameter description, see member data documentation.
    ConicGradientPattern(const Point& aCenter, Float aAngle, Float aStartOffset,
  ```

```
                          Float aEndOffset, GradientStops* aStops,
                          const Matrix& aMatrix = Matrix())
        : mCenter(aCenter),
          mAngle(aAngle),
          mStartOffset(aStartOffset),
          mEndOffset(aEndOffset),
          mStops(aStops),
          mMatrix(aMatrix) {}

    PatternType GetType() const override { return PatternType::CONIC_GRADIENT; }

    Point mCenter;       //!< Center of the gradient
    Float mAngle;        //!< Start angle of gradient
    Float mStartOffset;  // Offset of first stop
    Float mEndOffset;    // Offset of last stop
    RefPtr<GradientStops>
        mStops;      /**< GradientStops object for this gradient, this
                          should match the backend type of the draw target
                          this pattern will be used with. */
    Matrix mMatrix;  //!< A matrix that transforms the pattern into user space
};
```

- The corresponding pattern type in `gfx/2d/Types.h`:

```
 enum class PatternType : int8_t {
   COLOR,
   SURFACE,
   LINEAR_GRADIENT,
   RADIAL_GRADIENT,
+  CONIC_GRADIENT
 };
```

- The `gfxPattern` constructor for conic gradients in `gfx/thebes/gfxPattern.cpp`, called by the web painting code:

```
gfxPattern::gfxPattern(gfxFloat cx, gfxFloat cy, gfxFloat angle,
                       gfxFloat startOffset, gfxFloat endOffset)
    : mExtend(ExtendMode::CLAMP) {
  mGfxPattern.InitConicGradientPattern(Point(cx, cy), angle, startOffset,
                                       endOffset, nullptr);
}
```

- Code abstracting different backends represented as "draw targets". The implementation for the Skia draw target will be described in the next section.

4.5 Skia

In Firefox, Skia is the graphics backend used by default on Linux, Android, macOS and a very small subset of Windows devices [81]. It is a graphics library maintained by Google, other popular projects using it include Google Chrome, Chrome OS and Android [82].

Implementing conic gradients for Skia [80] mainly consisted of adding a case for conic gradient patterns in the `SetPaintPattern` method for `DrawTargetSkia.cpp`:

```
 static void SetPaintPattern(SkPaint& aPaint, const Pattern& aPattern,
                             Float aAlpha = 1.0,
                             const SkMatrix* aMatrix = nullptr,
                             const Rect* aBounds = nullptr) {
   switch (aPattern.GetType()) {
     // [...]
+    case PatternType::CONIC_GRADIENT: {
```

```
+      const ConicGradientPattern& pat =
+          static_cast<const ConicGradientPattern&>(aPattern);
+      GradientStopsSkia* stops =
+          static_cast<GradientStopsSkia*>(pat.mStops.get());
+      if (!stops || stops->mCount < 2 || !pat.mCenter.IsFinite() ||
+          !IsFinite(pat.mAngle)) {
+        aPaint.setColor(SK_ColorTRANSPARENT);
+      } else {
+        SkMatrix mat;
+        GfxMatrixToSkiaMatrix(pat.mMatrix, mat);
+        if (aMatrix) {
+          mat.postConcat(*aMatrix);
+        }
+
+        SkScalar cx = SkFloatToScalar(pat.mCenter.x);
+        SkScalar cy = SkFloatToScalar(pat.mCenter.y);
+
+        // Skia's sweep gradient angles are relative to the x-axis, not the
+        // y-axis.
+        Float angle = (pat.mAngle * 180.0 / M_PI) - 90.0;
+        if (angle != 0.0) {
+          mat.preRotate(angle, cx, cy);
+        }
+
+        SkTileMode mode = ExtendModeToTileMode(stops->mExtendMode, Axis::BOTH);
+        sk_sp<SkShader> shader = SkGradientShader::MakeSweep(
+            cx, cy, &stops->mColors.front(), &stops->mPositions.front(),
+            stops->mCount, mode, 360 * pat.mStartOffset, 360 * pat.mEndOffset,
+            0, &mat);
+
+        if (shader) {
+          aPaint.setShader(shader);
+        } else {
+          aPaint.setColor(SK_ColorTRANSPARENT);
+        }
+      }
+      break;
+    }
     // [...]
   }
 }
```

Like for other gradients:

- the parameters are passed in from the Moz2D API
- the box is painted transparent in case of invalid parameters or failure
- transformation matrices are converted and applied at this point (for CSS transforms)

As for the shader, `SkGradientShader::MakeSweep` was the right method according to the Skia documentation [83]. The initial Skia implementation used the following parameters:

```
sk_sp<SkShader> shader = SkGradientShader::MakeSweep(
    cx, cy, &stops->mColors.front(), &stops->mPositions.front(),
    stops->mCount, 0, &mat);
```

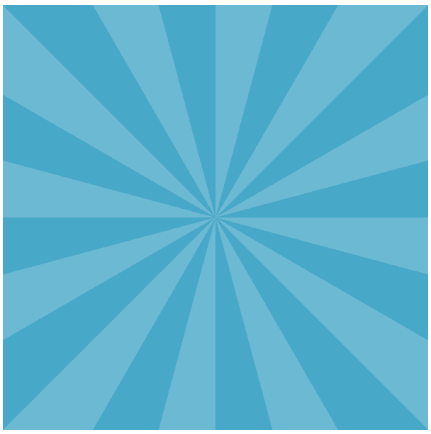
Unfortunately, while this worked well for many cases, two cases from the polyfill website were broken:

- `conic-gradient(red -50%, gold 110%)` rendered the same as `conic-gradient(red 0%, gold 100%)`.

(a) Expected

(b) Broken

- `repeating-conic-gradient` rendered the same as its `conic-gradient` counterpart.



(a) Expected

(b) Broken

To fix this issue, Blink's implementation of conic gradients was checked to see how Skia was used. Blink first calls `PaintShader::MakeSweepGradient` [84], which indirectly calls `SkGradientShader::MakeSweep` [85] [86]. However, it uses a 10-argument version that was not found on the Skia shader documentation [83], as opposed to the 7-argument version used in my initial implementation.

The extra arguments were:

- start offset: the position of the first color stop
- end offset: the position of the last color stop
- extend mode: this determines how the image is extended outside the bounds of the image. The same extend mode other gradient types used was chosen.

Similarly to WebRender, the start offset and the end offset were needed for Skia to properly render the shader. They needed to be converted from percentages to degrees, by multiplying the value by 360.

```
+       SkTileMode mode = ExtendModeToTileMode(stops->mExtendMode, Axis::BOTH);
        sk_sp<SkShader> shader = SkGradientShader::MakeSweep(
            cx, cy, &stops->mColors.front(), &stops->mPositions.front(),
-           stops->mCount, 0, &mat);
+           stops->mCount, mode, 360 * pat.mStartOffset, 360 * pat.mEndOffset,
+           0, &mat);
```

4.6 Web Platform Tests

Web Platform Tests (abbreviated as WPT) is a cross-browser test suite than runs on all major browser engines (Blink/Chromium, WebKit and Gecko). It contains tests contributed from Mozilla, Google, Apple, but also from independent contributors. The main aim of web platform tests to ensure that implementations by one browser engine are compatible with other browser engines and prevent web compatibility issues [87]. Tests can be contributed by sending a pull request to the upstream Github repository, but can also be contributed directly in the Chromium or Firefox source code, in which case they will be synced back to the upstream project by a bot.

In Firefox, web platform tests are setup in three parts:

- the metadata, consisting of INI files with preferences that need to be enabled and the tests that are expected to fail. The metadata is specific to the Firefox source code.
- the tests themselves, written in web technologies, which are synced with the Github repository
- the tooling, which takes care of synchronising and running the tests. It is not directly relevant to this project.

Tests automatically fail when the metadata is incorrect, e.g. when a test is marked as expected to fail, but actually passes, or vice-versa. They can be ran using the `./mach wpt` command in Firefox.

4.6.1 Updating the metadata

For conic gradients, three web platform tests were already written by external people:

- a `background-image` parsing test (`css/css-images/gradient/color-stops-parsing.html`), which tested that gradient functions with different parameters were correctly parsed in the `background-image` CSS property. This test was contributed by Florin Malita, a Google engineer. [88]
- a `border-image-source` serialisation test (`border-image-source-computed.sub.html`), testing the correctness of the `getComputedStyle()` serialisation for the `border-image-source` CSS property, which includes gradient values. This test was contributed by Eric Willigers, a Chromium contributor. [89]
- a rendering test (`css/css-images/multiple-position-color-stop-conic.html`), which tested that conic gradient with 2 hard color stops were rendered correctly. It was contributed by Fredrik Söderquist, at the time working at Opera. [90]

The metadata marked those three tests as failing. The implementation done for this project made those tests pass when ran with the conic gradient feature flag enabled. The metadata was updated to enable the feature flag for those tests instead of marking them as failing [91]. For instance:

```
[border-image-source-computed.sub.html]
  [Property border-image-source value 'conic-gradient(from 90deg at 80% 90%, lime, black)']
    expected: FAIL
```

was updated to:

```
[border-image-source-computed.sub.html]
  prefs: [layout.css.conic-gradient.enabled:true]
```

Early on during the implementation, the rendering test failed for WebRender only, due to color stop normalisation issues, this was fixed as described in the WebRender section.

Having tests contributed by external contributors start passing confirms the correctness of the implementation. However, one rendering test was not enough to test different edge cases of conic gradients, so more test cases were added.

4.6.2 Adding new tests

There was only one pre-existing rendering web platform test for conic gradients, compared to 14 rendering tests for linear gradients and 7 for radial gradients. It is reasonable that radial or conic gradients have less tests than linear gradients, since some linear gradient tests actually test shared code between all three gradient types. Tests were written and merged to test different edge cases of conic gradients. Most

were based on wrench tests written earlier on in the project for WebRender, but used HTML and CSS instead of YAML files.

Here is an example from the `normalization-conic.html` test:

```
<!doctype html>
<meta charset="utf-8">
<title>Conic gradient stop normalization</title>
<link rel="help" href="https://drafts.csswg.org/css-images-4/#conic-gradients">
<meta name="assert" content="Rendering of conic-gradient with normalized color stops">
<link rel="match" href="reference/100x100-blue.html">
<style>
  #gradient {
    width: 100px;
    height: 100px;
    background-image: conic-gradient(green -50%, blue -50%);
  }
</style>
<div id="gradient"></div>
```

A web platform test is simply an HTML file:

- The HTML doctype and the charset meta tag are not necessary for conic gradients, but it is good practice to include them in any web page.
- The help link tag links to the specification
- The assert meta tag describes what is being tested
- The match link tag links to the matching reference file, in this case, rendering a blue square:

  ```
  <!doctype html>
  <div style="width: 100px;height: 100px;background-color: blue;"></div>
  ```

The complete commit adding the tests can be found on Phabricator [92] and the tests can be categorised as such:

- Tests for conic gradients in a basic configuration.

  ```
  testing/web-platform/tests/css/css-images/conic-gradient-angle-negative.html
  testing/web-platform/tests/css/css-images/conic-gradient-angle.html
  testing/web-platform/tests/css/css-images/conic-gradient-center.html
  testing/web-platform/tests/css/css-images/multiple-position-color-stop-conic-2.html
  testing/web-platform/tests/css/css-images/repeating-conic-gradient.html
  testing/web-platform/tests/css/css-images/tiled-conic-gradients.html
  ```

- Tests for gradients with color stops outside the range of 0 to 100%, since both the Skia and WebRender had related bugs early on the implementation as mentioned previously.

  ```
  testing/web-platform/tests/css/css-images/normalization-linear-2.html
  testing/web-platform/tests/css/css-images/normalization-linear.html
  testing/web-platform/tests/css/css-images/normalization-radial-2.html
  testing/web-platform/tests/css/css-images/normalization-radial.html
  testing/web-platform/tests/css/css-images/normalization-conic-2.html
  testing/web-platform/tests/css/css-images/normalization-conic.html
  testing/web-platform/tests/css/css-images/out-of-range-color-stop-conic.html
  ```

- Tests for the degenerate case, where a repeating gradient has two stops at the same position (e.g. `repeating-conic-gradient(orange 50%, blue 50%)`), in which case the element should be completely filled with the second color (blue).

  ```
  testing/web-platform/tests/css/css-images/normalization-linear-degenerate.html
  testing/web-platform/tests/css/css-images/normalization-radial-degenerate.html
  testing/web-platform/tests/css/css-images/normalization-conic-degenerate.html
  ```

4.6.3 Interoperability of tests

When adding new web platform tests, it is important to check whether the references for the tests are correct and interoperable with other web browsers.

Having the tests reviewed by an experienced Gecko engineer and merged [92] is usually enough to check the correctness, but it only guarantees that the tests pass in Firefox. It is possible that mistakes are overlooked, which is why it is worth checking how the test results for other browsers.

To do this, there is a website called wpt.fyi, which is a dashboard showing results from web platform test runs for major browsers. It is worth noting that wpt.fyi does not take in account the test metadata from different browsers, meaning that feature flags will not be turned on, which causes Firefox results to be marked as failed.

The run tested the latest alpha versions of each browser and can be found at [93]. The results can be found in the screenshot below:

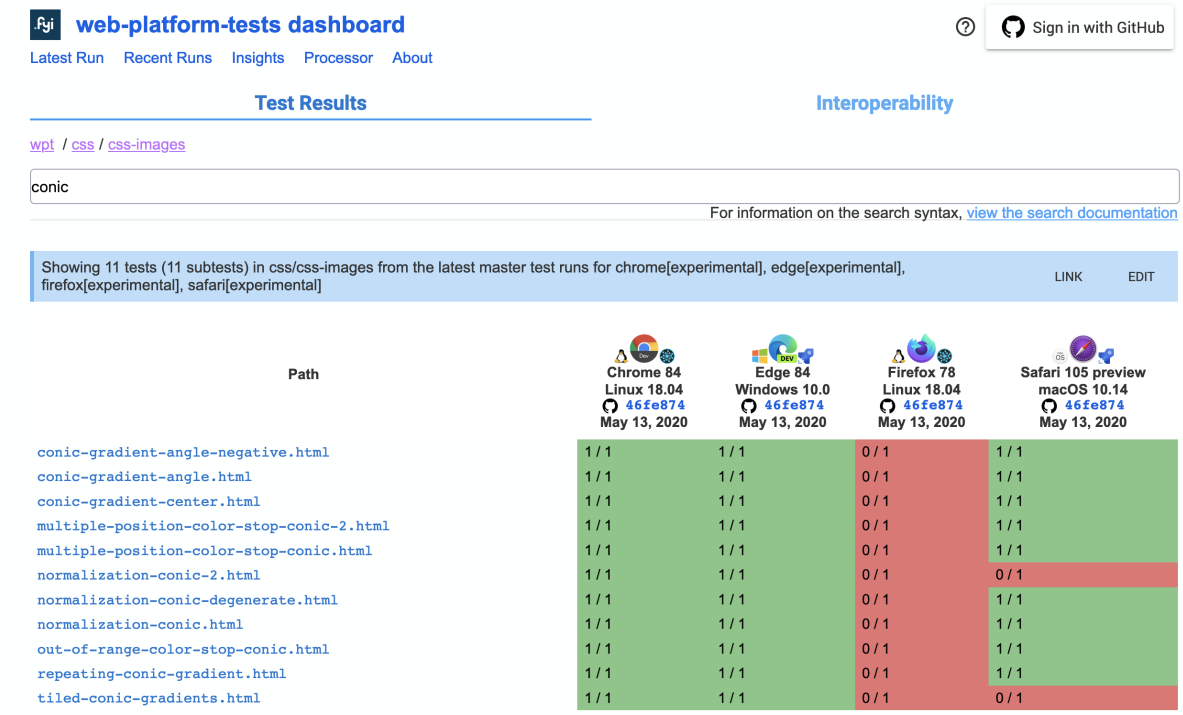
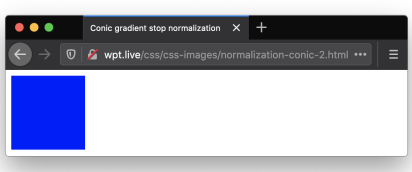


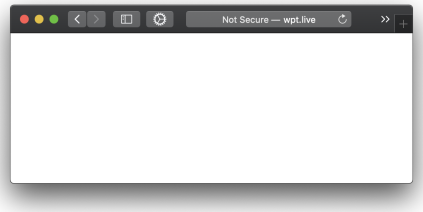
Figure 4.9: wpt.fyi results for conic gradient related tests

As seen in the screenshot above, all conic gradient tests are passing in Chromium and most of them pass in Safari (which uses WebKit).

It turns out the `normalization-conic-2.html` test (written part of this project) reveals a bug in WebKit's implementation, which is why the test was failing. This test uses an element filled with `conic-gradient(blue 150%, red 150%)` which should render a blue square. However, Safari renders a transparent square:



(a) Expected (Chromium and Firefox)



(b) Broken (Safari)

The other failing test, `tiled-conic-gradients.html`, is a small rendering glitch, that is not seen when ran live on Safari, so it is safe to assume that this may be related to the hardware used by wpt.fyi:

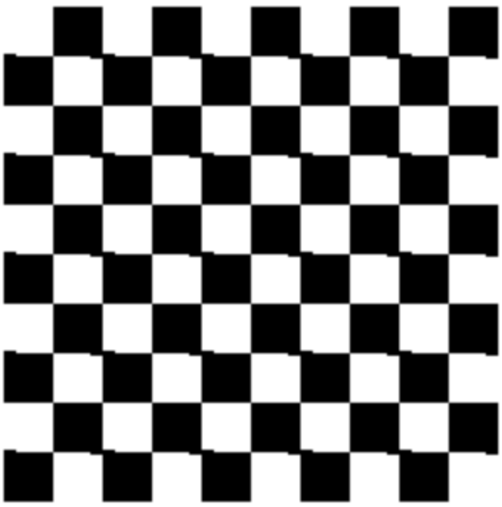


Figure 4.11: Graphical glitch on tiled-conic-gradients.html

4.7 Developer Tools support

Like other browsers, Firefox has developer tools for web developers to debug their webpages. The developer tool requiring extra work for this project is the inspector tool (also named the elements tool in other web browsers). This tool allows inspecting the structure of the webpage and displays a sidebar which shows the CSS styles for the selected element.

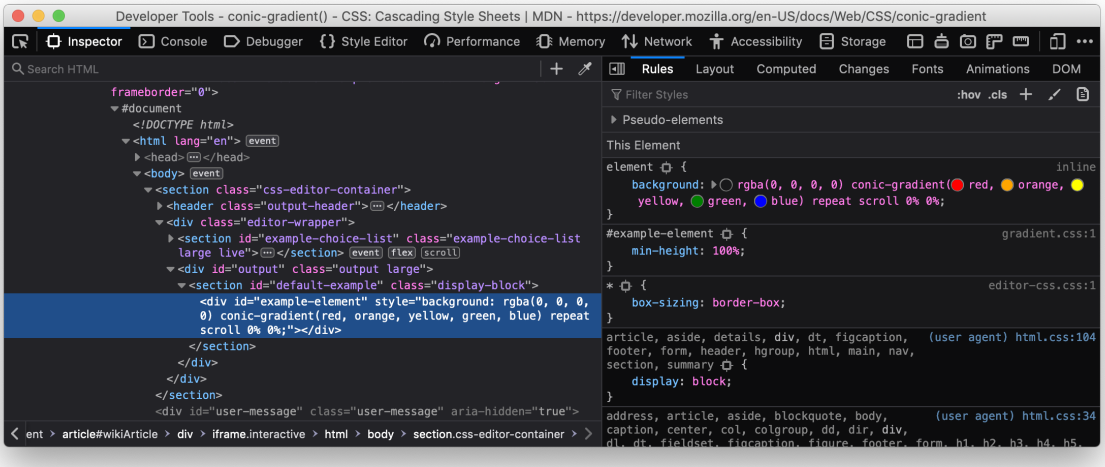


Figure 4.12: Firefox Inspector tool

The inspector works with and understands the newly added CSS feature without any extra effort. However, some bits of the inspector need extra work to provide an optimal experience for developers.

4.7.1 Auto-completion

To make authoring faster, tools usually suggest keywords inside a popup, so the user can simply type a part of a keyword and select it to get the whole keyword. The inspector is no exception:

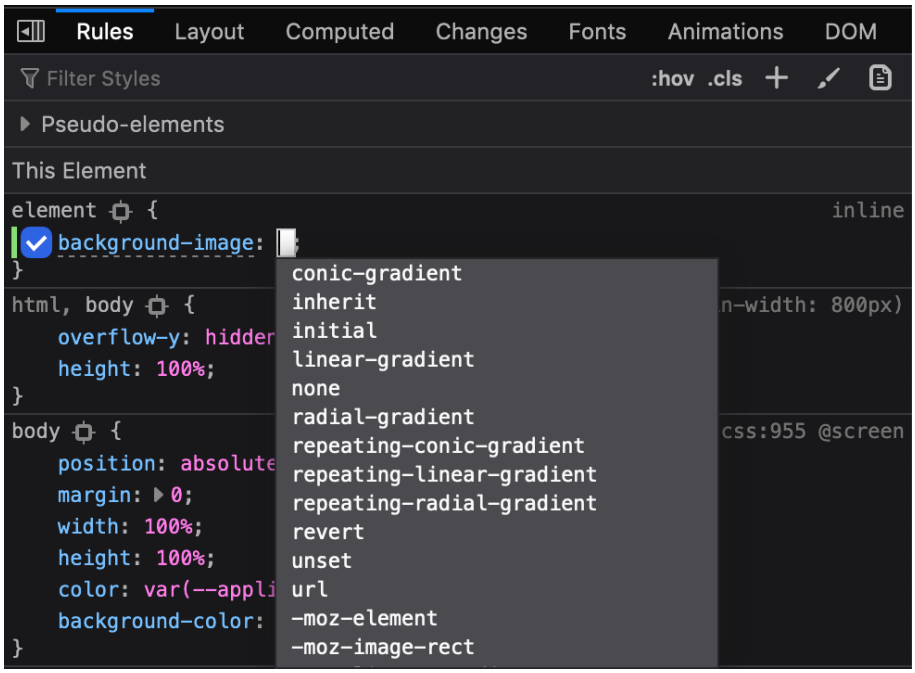


Figure 4.13: Auto-completion in the Rules view

To add conic gradients to the autocompletion list, the existing `collect_completion_keywords` function in the style system was extended:

```
 fn collect_completion_keywords(f: KeywordsCollectFn) {
     f(&[
         // [...]
     ]);

+    if static_prefs::pref!("layout.css.conic-gradient.enabled") {
+        f(&[
+            "conic-gradient",
+            "repeating-conic-gradient",
+        ]);
+    }
 }
```

4.7.2 Color and angle swatches

For convenience and to make the developer tool require less technical knowledge to use, Firefox has added swatches for colors and angles. Clicking the color swatch brings up a color picker, and clicking it while holding the “Shift” key switches between color formats. As for the angle swatch, clicking it while holding “Shift” switches between different angle units.

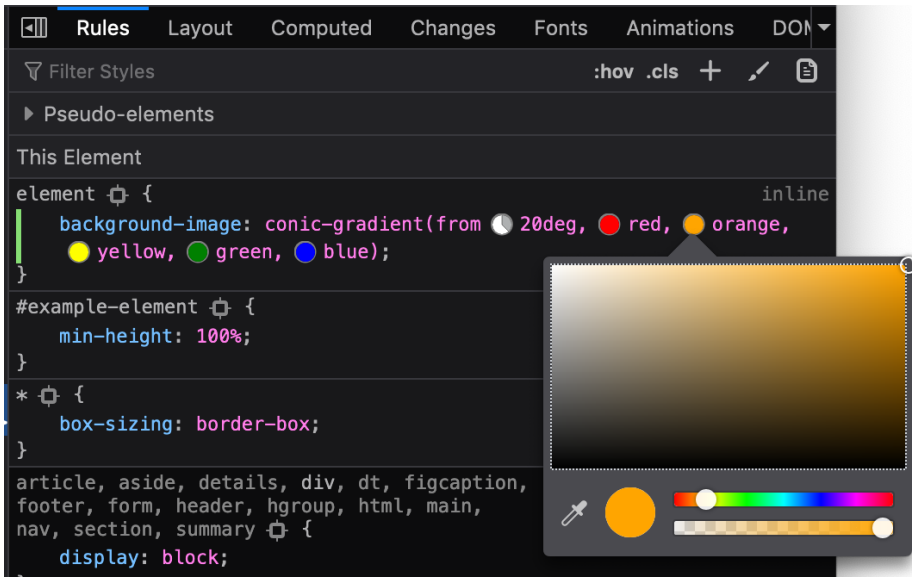


Figure 4.14: Swatches in the Rules view

Those are already shown for other CSS functions using angles or colors, but were not initially shown for conic gradients. To fix this, cases were added to the developer tools output parser (in `devtools/client/shared/output-p`

```
const CONIC_GRADIENT_ENABLED = Services.prefs.getBoolPref(
  "layout.css.conic-gradient.enabled"
);

// Functions that accept an angle argument.
const ANGLE_TAKING_FUNCTIONS = [
  // [...]
  ...(CONIC_GRADIENT_ENABLED
    ? ["conic-gradient", "repeating-conic-gradient"]
    : []),
  // [...]
];

// Functions that accept a color argument.
const COLOR_TAKING_FUNCTIONS = [
  // [...]
  ...(CONIC_GRADIENT_ENABLED
    ? ["conic-gradient", "repeating-conic-gradient"]
    : []),
  "drop-shadow",
];
```

4.8 HTML canvas API

The HTML `<canvas>` element enables drawing graphics and animations using the canvas JavaScript API (known as `CanvasRenderingContext2D`). It may be used in cases where rendering multiple HTML elements styled with CSS may be inefficient, for instance games where high performance may be essential.

There are existing `createLinearGradient` and `createRadialGradient` functions that are in the HTML specification, but there is not currently any `createConicGradient` function. I have filed an issue on the specification Github repository and there is some early work-in-progress specification work being done in collaboration with the Google Chrome team [94]. The current function definition decided on is:

```
interface mixin CanvasFillStrokeStyles {
```

```
  ...
  CanvasGradient createConicGradient(double angle, double cx, double cy);
};
```

The `angle` parameter is analogous to the one in the CSS function, while the `cx` and `cy` parameters represent the position/center parameter from the CSS function.

Here is an example usage from the work-in-progress proposal [95]:

```
const canvas = document.createElement("canvas");
const ctx = canvas.getContext("2d");

const grad = ctx.createConicGradient(0, 100, 100);

grad.addColorStop(0, "red");
grad.addColorStop(0.25, "orange");
grad.addColorStop(0.5, "yellow");
grad.addColorStop(0.75, "green");
grad.addColorStop(1, "blue");

ctx.fillStyle = grad;
ctx.fillRect(0, 0, 200, 200);
document.body.append(canvas);
```

For this project, an implementation has been made in the Firefox rendering engine [96]. However, it has not been merged and tests have not yet been written, since it is awaiting for the final specification. The code for the canvas API is located in the `dom/canvas/` directory. The function declaration was added to the `WebIDL` file, which defines the JavaScript methods exposed to the web. The rest of the implementation was analogous to the existing radial or linear gradient counterparts, adjusted for the different conic gradient parameters and re-used the CSS `conic-gradient` function graphics code.

4.9 Future work

4.9.1 Direct2D

One remaining part of this project is the implementation for the Direct2D backend. This is the only remaining bit of work needed to enable conic gradients by default for all users. Direct2D is used on Windows when WebRender is not supported by the underlying hardware. In rare cases where Direct2D is not supported, Skia is used as a fallback [81]. Skia could technically be used on all Windows machines, as the library supports it, but Direct2D was chosen by the graphics team since it has a better performance due to being designed specifically for Windows by Microsoft.

For Direct2D, an untested work in progress commit developed on macOS was submitted [97] and will be described here. Conic gradients on Direct2D can be done using the mesh gradient library methods, however those methods are only available on Windows 10 and above [98], so they cannot be used in Firefox which has to support Windows 7 and 8. However, a solution that works on all versions of Windows would be creating a custom effect (similar to shaders) and rendering it. The work in progress commit uses this approach, based on the WebRender shader and the existing radial gradient code, but it cannot be tried out and finished without a Windows development environment.

4.9.2 Cairo

Implementing conic gradients for printing was deemed low priority by the Firefox Layout team because the `background-image` CSS property, which is the main use-case for conic gradients, is disabled by default when printing web pages for readability and can only be enabled with an internal preference [99]. Given this fact, implementing printing support is not essential to enable conic gradient by default.

However, if this were to be done, a possible approach would be to use Skia to render the conic gradient into a surface and pass that surface to DrawTargetCairo for printing. An alternative approach would be using the Cairo APIs to draw the gradient, it is more complex but provides greater rendering performance, which does not matter much with printing.

Chapter 5

Conclusion

Working on a 20-year old codebase like Firefox is very challenging, especially since the development process is mostly made up of in-house tools: from finding issues, installing dependencies, building Firefox, getting around the source code, to getting the change committed, reviewed, tested and merged.

The web is also incredibly complex and the underlying rendering engines are even more so. This project has shown that adding a single CSS feature involves various topics: software engineering, software architecture, parsing or computer graphics. It also requires a lot of care and collaboration regarding how developers may use the technology, as web compatibility and interoperability concerns show.

Despite these complex challenges, an implementation of conic gradients was made for the Skia and WebRender backends, which cover almost all platforms: Android, macOS, Linux and a minority of Windows devices. This implementation is available in Firefox 75 (released on April 8th), behind the `layout.css.conic-gradient.enabled` flag, as shown on the Can I Use? website [1]:

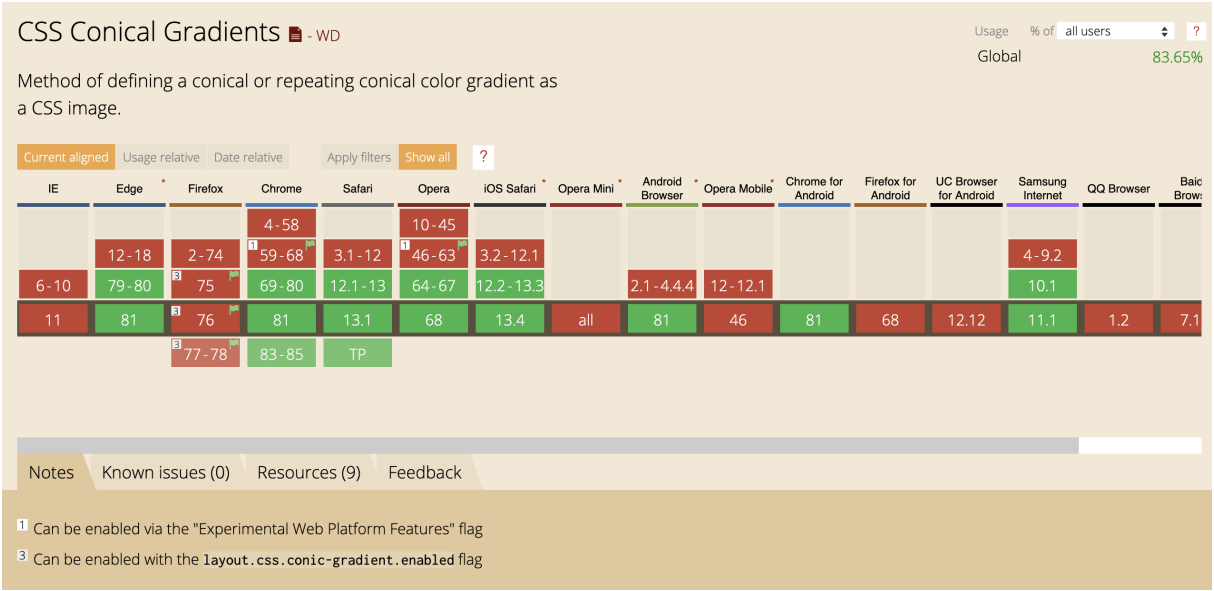


Figure 5.1: Screenshot of Can I Use? as of May 6th

The interoperability of the implementation was evidenced by the contribution of new web platform tests, which also run in the Chromium and WebKit repositories. These tests also contribute to making other implementations more interoperable, as evidenced by an issue found in WebKit's implementation.

However, there are few things that were not achieved and could be potential future projects:

- The implementation for the Cairo backend, only used for printers, is missing, due to the complexity of the backend. However, it is less important, since background images are disabled for printing unless a hidden preference is enabled, as mentioned in the Cairo section.
- The implementation for the Direct2D backend, used on the majority of Windows devices, has not been finished yet because of the lack of Windows development environment. With a Direct2D implementation done, Firefox could enable the conic gradient flag by default and catch up with the competition, as it was the only major browser to not implement the feature since January 2020.
- The current implementation is not fully-standard compliant yet as it does not allow mixing angles and percentages in `calc()` expressions. This is however less important for interoperability since no other web browser currently supports it.
- In retrospect, writing conic gradient tests for `border-image` would have been a good idea, since the web painting code uses a different code path for this case.
- An interesting fact was that conic gradients did not have an equivalent JavaScript API method for the HTML `<canvas>` element, since they are newer, despite the presence of equivalent methods for linear and radial gradients. A working implementation has been done but not merged. Contributing the API to the specification and adding a well-tested implementation to Firefox could be an interesting future potential project.

Regarding the Cairo and Direct2D backends, it is worth noting that those two implementations may not be necessary in the long term, since Mozilla is aiming to replace those backends with WebRender.

Bibliography

[1] CSS Conical Gradients - Can I Use. [Online]. Available: https://caniuse.com/#feat=css-conic-gradients

[2] L. Verou. Conical gradients, today! [Online]. Available: http://lea.verou.me/2015/06/conical-gradients-today/

[3] ——. About me. [Online]. Available: http://lea.verou.me/about/

[4] H. W. Lie and B. Bos. (1996) Cascading Style Sheets, level 1. [Online]. Available: https://www.w3.org/TR/CSS1/

[5] C. L. Håkon Wium Lie, Bert Bos and I. Jacobs. (1998) Cascading Style Sheets, level 2. [Online]. Available: https://www.w3.org/TR/2008/REC-CSS2-20080411/

[6] H. W. Lie and B. Bos. (2011) Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) specification. [Online]. Available: https://www.w3.org/TR/2011/PR-CSS2-20110412/

[7] WHATWG. FAQ — WHATWG. [Online]. Available: https://whatwg.org/faq#what-is-the-whatwg

[8] W3C. (2019) About W3C. [Online]. Available: https://www.w3.org/Consortium/

[9] J. Jaffe. W3C and WHATWG to work together to advance the open web platform. [Online]. Available: https://www.w3.org/blog/2019/05/w3c-and-whatwg-to-work-together-to-advance-the-open-web-platform/

[10] W3C. (2019) World Wide Web Consortium process document. [Online]. Available: https://www.w3.org/2019/Process-20190301/#maturity-levels

[11] ——. (2017) CSS Image Values and Replaced Content Module Level 4. [Online]. Available: https://www.w3.org/TR/2017/WD-css-images-4-20170413/

[12] StackExchange. Why is JavaScript disabled in the Tor Browser Bundle? [Online]. Available: https://security.stackexchange.com/questions/40620/why-is-javascript-disabled-in-the-tor-browser-bundle

[13] History - Badminton - wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Badminton#History

[14] Quirks Mode and Standards Mode — mdn. [Online]. Available: https://developer.mozilla.org/docs/Web/HTML/Quirks_Mode_and_Standards_Mode

[15] WHATWG. (2020) Quirks mode. [Online]. Available: https://quirks.spec.whatwg.org/

[16] StatCounter. Top 5 Desktop, Tablet & Console Browsers from July 2008 - Sept 2016. [Online]. Available: https://gs.statcounter.com/#browser-ww-monthly-200807-201609

[17] D. Hyatt. Introducing CSS gradients — webkit. [Online]. Available: https://webkit.org/blog/175/introducing-css-gradients/

[18] S. Fraser. CSS3 gradients — WebKit. [Online]. Available: https://webkit.org/blog/1424/css3-gradients/

[19] CSS gradients - Can I Use. [Online]. Available: https://caniuse.com/#feat=css-gradients

[20] Compatibility Standard. [Online]. Available: https://compat.spec.whatwg.org/#css-gradients-webkit-linear-gradient

[21] FollowAndrew. (2019) CSS conic gradient effects tutorial. [Online]. Available: https://followandrew.dev/css-conic-gradient-effects-tutorial/

[22] Alligator.io. Introduction to conic gradients in CSS ← Alligator.io. [Online]. Available: https://alligator.io/css/conic-gradients/

[23] Twitter. Search for "conic gradient". [Online]. Available: https://twitter.com/search?q=conic%20gradient&src=typed_query&f=live

[24] L. Verou. (2015) CSS conic-gradient() polyfill. [Online]. Available: https://leaverou.github.io/conic-gradient/

[25] Github. Search for the conic gradient polyfill. [Online]. Available: https://github.com/search?l=HTML&p=1&q=https%3A%2F%2Fleaverou.github.io%2Fconic-gradient%2F&type=Code

[26] T. A. Jr., E. J. Etemad, and L. Verou. (2020) CSS Images Level 4 specification (latest version). [Online]. Available: https://drafts.csswg.org/css-images-4/#conic-gradients

[27] Bugzilla. [Online]. Available: https://bugzilla.mozilla.org

[28] Codetribute. [Online]. Available: https://codetribute.mozilla.org

[29] Mercurial. [Online]. Available: https://www.mercurial-scm.org/

[30] Build firefox! - building Firefox for macOS. [Online]. Available: https://developer.mozilla.org/docs/Mozilla/Developer_guide/Build_Instructions/Mac_OS_X_Prerequisites#Build_Firefox!

[31] Building Firefox for Windows - Mozilla — MDN. [Online]. Available: https://developer.mozilla.org/docs/Mozilla/Developer_guide/Build_Instructions/Windows_Prerequisites

[32] Searchfox. [Online]. Available: https://searchfox.org

[33] Phacility - Phabricator. [Online]. Available: https://phacility.com/phabricator/

[34] Phabricator. [Online]. Available: https://phabricator.services.mozilla.com/

[35] Mochitest — MDN. [Online]. Available: https://developer.mozilla.org/docs/Mozilla/Projects/Mochitest

[36] Reftests - Firefox Source Docs. [Online]. Available: https://firefox-source-docs.mozilla.org/web-platform/writing-tests/reftests.html

[37] Fuzzy matching - reftests - Firefox Source Docs. [Online]. Available: https://firefox-source-docs.mozilla.org/web-platform/writing-tests/reftests.html#fuzzy-matching

[38] Mozilla. Reftest analyser. [Online]. Available: https://hg.mozilla.org/mozilla-central/raw-file/tip/layout/tools/reftest/reftest-analyzer.xhtml

[39] Try server - firefox source docs. [Online]. Available: https://firefox-source-docs.mozilla.org/tools/try/index.html

[40] Configuring try - firefox source docs. [Online]. Available: https://firefox-source-docs.mozilla.org/tools/try/configuration.html

[41] Treeherder. [Online]. Available: https://treeherder.mozilla.org

[42] Firefox Nightly. [Online]. Available: https://nightly.mozilla.org

[43] Firefox Release Calendar - MozillaWiki. [Online]. Available: https://wiki.mozilla.org/Release_Management/Calendar

[44] Noel. The Rust Language — Lambda the Ultimate. [Online]. Available: http://lambda-the-ultimate.org/node/4009

[45] Rust Programming Language. [Online]. Available: https://www.rust-lang.org/

[46] L. Clark. Inside a super fast CSS engine: Quantum CSS (aka Stylo). [Online]. Available: https://hacks.mozilla.org/2017/08/inside-a-super-fast-css-engine-quantum-css-aka-stylo/

[47] LearnOpenGL - Shaders. [Online]. Available: https://learnopengl.com/Getting-started/Shaders

[48] webrender/README.md at 12dce8e771720d753aeab539100ffa7dcaddf705 · servo/webrender. [Online]. Available: https://github.com/servo/webrender/blob/12dce8e771720d753aeab539100ffa7dcaddf705/README.md

[49] GLSL Tutorial - Fragment Shader - Lighthouse3d.com. [Online]. Available: http://www.lighthouse3d.com/tutorials/glsl-tutorial/fragment-shader/

[50] GLSL Tutorial - Vertex Shader - Lighthouse3d.com. [Online]. Available: http://www.lighthouse3d.com/tutorials/glsl-tutorial/vertex-shader/

[51] OpenGL - Drawing polygons. [Online]. Available: https://open.gl/drawing

[52] Servo, the parallel browser engine. [Online]. Available: https://servo.org/

[53] W3C. Specified Value - Assigning property values, Cascading, and Inheritance. [Online]. Available: https://www.w3.org/TR/CSS22/cascade.html#specified-value

[54] ——. Computed Value - Assigning property values, Cascading, and Inheritance. [Online]. Available: https://www.w3.org/TR/CSS22/cascade.html#computed-value

[55] ——. CSS Object Model (CSSOM). [Online]. Available: https://drafts.csswg.org/cssom/#resolved-values

[56] ——. Image Sources: the background-image property - CSS Backgrounds and Borders Module Level 3. [Online]. Available: https://www.w3.org/TR/css-backgrounds-3/#the-background-image

[57] T. Nguyen. D62158 - Bug 1614160 - Add AngleOrPercentage to style system. r=emilio. [Online]. Available: https://phabricator.services.mozilla.com/D62158

[58] W3C. Mixing percentages and dimensions - CSS Values and Units Module Level 3. [Online]. Available: https://www.w3.org/TR/css-values-3/#mixed-percentages

[59] [css-values] Ambiguities with 0 valid for all dimensions · Issue #1162 · w3c/csswg-drafts. [Online]. Available: https://github.com/w3c/csswg-drafts/issues/1162

[60] conic-gradient() syntax - CSS Images Module Level 4. [Online]. Available: https://drafts.csswg.org/css-images-4/#valdef-conic-gradient-angle

[61] Generics - Rust By Example. [Online]. Available: https://doc.rust-lang.org/rust-by-example/generics.html

[62] W3C. Color stop lists - CSS Images Module Level 4. [Online]. Available: https://drafts.csswg.org/css-images-4/#color-stop-syntax

[63] T. Nguyen. D62544 - Bug 1614648 - Make GradientItem and ColorStop support angular color stops. r=emilio. [Online]. Available: https://phabricator.services.mozilla.com/D62544

[64] ——. D62923 - Bug 1615489 - Refactor GenericGradient for conic-gradient support. r=emilio. [Online]. Available: https://phabricator.services.mozilla.com/D62923

[65] Serialization - CSS Images Module Level 4. [Online]. Available: https://drafts.csswg.org/css-images-4/#serialization

[66] gradient-position-valid.html - mozsearch. [Online]. Available: https://searchfox.org/mozilla-central/rev/fa52bedc4b401c12251513fa1c9df1753a29abb2/testing/web-platform/tests/css/css-images/parsing/gradient-position-valid.html#16

[67] D67461 - Bug 1618997 - Omit center positions in conic/radial gradient serialization. [Online]. Available: https://phabricator.services.mozilla.com/D67461

[68] W3C. CSS Values and Units Module Level 3. [Online]. Available: https://drafts.csswg.org/css-values-3/#position

[69] Platform/GFX/WebRender Where - MozillaWiki. [Online]. Available: https://wiki.mozilla.org/Platform/GFX/WebRender_Where

[70] StaticPrefList.yaml - mozsearch. [Online]. Available: https://searchfox.org/mozilla-central/rev/a707541ff423ade0d81cef6488e6ecfa09273886/modules/libpref/init/StaticPrefList.yaml#3979

[71] T. Nguyen. D61599 - Bug 1614890 - Implement conic-gradient for WebRender graphics backend. [Online]. Available: https://phabricator.services.mozilla.com/D61599

[72] Wikipedia. Geometric primitive. [Online]. Available: https://en.wikipedia.org/wiki/Geometric_primitive

[73] glprogramming.com. OpenGL Programming Guide. [Online]. Available: https://www.glprogramming.com/red/chapter07.html

[74] brush_conic_gradient.glsl - mozsearch. [Online]. Available: https://searchfox.org/mozilla-central/rev/0688ffdef223dac527c2fcdb25560118c4e4df51/gfx/wr/webrender/res/brush_conic_gradient.glsl

[75] Vectors - Data Type (GLSL) - OpenGL Wiki. [Online]. Available: https://www.khronos.org/opengl/wiki/Data_Type_(GLSL)#Vectors

[76] T. Nguyen. D65391 - Bug 1616255 - Handle start and end offsets in conic-gradient WR shader. [Online]. Available: https://phabricator.services.mozilla.com/D65391

[77] Debugging WebRender · servo/webrender Wiki. [Online]. Available: https://github.com/servo/webrender/wiki/Debugging-WebRender#wrench

[78] T. Nguyen. D63213 - Bug 1616106 - Add more tests and reference images for conic-gradient wrench test suite. [Online]. Available: https://phabricator.services.mozilla.com/D63213

[79] Platform/GFX/Moz2D - MozillaWiki. [Online]. Available: https://wiki.mozilla.org/Platform/GFX/Moz2D

[80] T. Nguyen. D63415 - Bug 1616587 - Implement conic-gradient for Skia graphics backend. [Online]. Available: https://phabricator.services.mozilla.com/D63415

[81] GfxInfo.cpp - mozsearch. [Online]. Available: https://searchfox.org/mozilla-central/rev/a707541ff423ade0d81cef6488e6ecfa09273886/widget/windows/GfxInfo.cpp#1447

[82] Skia Graphics Library. [Online]. Available: https://skia.org/

[83] SkPaint Overview. [Online]. Available: https://skia.org/user/api/skpaint_overview

[84] chromium/gradient.cc at c20d681c9c067c4e15bb1408f17114b9e8cba294 · chromium/chromium. [Online]. Available: https://github.com/chromium/chromium/blob/c20d681c9c067c4e15bb1408f17114b9e8cba294/third_party/blink/renderer/platform/graphics/gradient.cc#L327

[85] chromium/paint_shader.cc at 2f778d2bd09579100bb1f1de5ab666056c1fd48b · chromium/chromium. [Online]. Available: https://github.com/chromium/chromium/blob/2f778d2bd09579100bb1f1de5ab666056c1fd48b/cc/paint/paint_shader.cc#L174

[86] chromium/paint_shader.cc at 2f778d2bd09579100bb1f1de5ab666056c1fd48b · chromium/chromium. [Online]. Available: https://github.com/chromium/chromium/blob/2f778d2bd09579100bb1f1de5ab666056c1fd48b/cc/paint/paint_shader.cc#L434

[87] web-platform-tests/wpt: Test suites for Web platform specs — including WHATWG, W3C, and others. [Online]. Available: https://github.com/web-platform-tests/wpt#readme

[88] Add a gradient color-stops parsing test by fmalita · Pull Request #10274 · web-platform-tests/wpt. [Online]. Available: https://github.com/web-platform-tests/wpt/pull/10274

[89] CSS: Test computed value of border properties (Ie074a5f3) · Gerrit Code Review. [Online]. Available: https://chromium-review.googlesource.com/c/chromium/src/+/1655588

[90] Add basic tests for multiple position gradient stops (I56bc9b43) · Gerrit Code Review. [Online]. Available: https://chromium-review.googlesource.com/c/chromium/src/+/1228120

[91] T. Nguyen. D63772 - Bug 1617397 - Update WPT expectations for conic-gradient. r=jgraham. [Online]. Available: https://phabricator.services.mozilla.com/D63772

[92] ——. D65928 - Bug 1616986 - Add WPT reftests for conic-gradient and stop normalization. r=emilio. [Online]. Available: https://phabricator.services.mozilla.com/D65928

[93] web-platform-tests dashboard. [Online]. Available: https://wpt.fyi/results/css/css-images?label=master&label=experimental&product=chrome-84.0.4143.2%20dev&product=edge-84.0.508.0&product=firefox-78.0a1&product=safari-105%20preview&aligned&q=conic

[94] T. Nguyen. Proposal: add createConicGradient to CanvasRenderingContext2D · Issue #5431 · whatwg/html. [Online]. Available: https://github.com/whatwg/html/issues/5431

[95] F. S. Aaron Krajeski and T. Nguyen. canvas2d/conic-gradient.md at master · fserb/canvas2d. [Online]. Available: https://github.com/fserb/canvas2D/blob/master/spec/conic-gradient.md

[96] T. Nguyen. D69465 - Bug 1627014 - Implement CanvasRenderingContext2D.createConicGradient. [Online]. Available: https://phabricator.services.mozilla.com/D69465

[97] ——. D66305 - Bug 1617396 - Implement conic-gradient for Direct2D graphics backend. [Online]. Available: https://phabricator.services.mozilla.com/D66305

[98] ID2D1DeviceContext2::CreateGradientMesh (d2d1_3.h) - Win32 apps — Microsoft Docs. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/api/d2d1_3/nf-d2d1_3-id2d1devicecontext2-creategradientmesh

[99] nsPrintSettingsImpl.cpp - mozsearch. [Online]. Available: https://searchfox.org/mozilla-central/rev/8bc4e35c9bb47c1fe3131e6155d9f482e1efef9a/widget/nsPrintSettingsImpl.cpp#22

[100] T. A. Jr. and E. J. Etemad. (2020) CSS Values Level 4 specification (latest version). [Online]. Available: https://drafts.csswg.org/css-values-4/#typedef-angle-percentage

[101] A. Tudor. Metallic button. [Online]. Available: https://codepen.io/thebabydino/pen/qdPRpN/

[102] W3C. Placing color stops - CSS Images Module Level 4. [Online]. Available: https://drafts.csswg.org/css-images-4/#conic-color-stops

Appendix A

Conic Gradient syntax

A.1 Conic gradient type

The syntax for a conic gradient as defined in the specification [26] is:

```
conic-gradient() = conic-gradient(
  [ from <angle> ]? [ at <position> ]?,
  <angular-color-stop-list>
)

<angular-color-stop-list> =
  <angular-color-stop> , [ <angular-color-hint>? , <angular-color-stop> ]#
<angular-color-stop> = <color> && <color-stop-angle>?
<angular-color-hint> = <angle-percentage>
<color-stop-angle> = <angle-percentage>{1,2}

<color-stop> = <color-stop-length> | <color-stop-angle>
```

A.2 <angle-percentage> type

In the CSS Values specification [100], `<angle-percentage>` is defined as:

> Equivalent to `[ <angle> | <percentage> ]`, where the `<percentage>` will resolve to an `<angle>`.

Also quoted from that last specification, `<angle>` is defined as a number followed by one of these units:

- `deg`

  Degrees. There are 360 degrees in a full circle.

- `grad`

  Gradians, also known as "gons" or "grades". There are 400 gradians in a full circle.

- `rad`

  Radians. There are 2π radians in a full circle.

- `turn`

  Turns. There is 1 turn in a full circle.

while `<percentage>` is a number followed by a percentage sign.

A.2.1 Examples

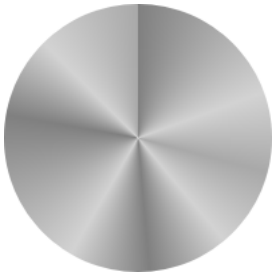
The following examples are equivalent ways of expressing the same `<angle-percentage>` value:

- `360deg`
- `1turn`
- `100%`
- `400grad`

These can be combined within a `calc()` expression, like so:

- `calc(360deg - 0.5turn)`
- `calc(100% - 50%)`

While this is not very useful in its current form, since `50%` could be written directly, this is useful for use with CSS variables. The following example is adapted from Ana Tudor's metallic button demo [101]:



```
div {
    --step: 35deg;
    background: conic-gradient(
        gray,
        darkgray  var(--step),
        lightgray calc(var(--step) * 2),
        darkgray  calc(var(--step) * 3),
        lightgray calc(var(--step) * 4),
        gray      calc(var(--step) * 5),
        lightgray calc(var(--step) * 6),
        darkgray  calc(var(--step) * 7),
        gray      calc(var(--step) * 8),
        lightgray calc(var(--step) * 9),
        darkgray
    );
    width: 100px;
    height: 100px;
    border-radius: 100%;
}
```

Figure A.1: Simplified metallic button demo

The specification allows mixing angles and percentages in the same in the same expression [58], but no web browser currently supports it. The use-case is low, since percentages in this case are equivalent to angles anyway. For instance, `calc(100% - 35deg)` could be written as `calc(1turn - 35deg)`, since the former is not supported.

Appendix B

Placing color stops

Quoted from the "Placing color stops" section of the specification [102]:

> Color stops are placed on a gradient line that curves around the gradient center in a circle, with both the `0%` and `100%` locations at `0deg`. Just like linear gradients, `0deg` points to the top of the page, and increasing angles correspond to clockwise movement around the circle.
>
> Note: It may be more helpful to think of the gradient line as forming a spiral, where only the segment from `0deg` to `360deg` is rendered. This avoids any confusion about "overlap" when you have angles outside of the rendered region.
>
> A color-stop can be placed at a location before `0%` or after `100%`; though these regions are never directly consulted for rendering, color stops placed there can affect the color of color-stops within the rendered region through interpolation or repetition (see repeating gradients). For example, `conic-gradient(red -50%, yellow 150%)` produces a conic gradient that starts with a reddish-orange color at `0deg` (specifically, `#f50`), and transitions to an orangish-yellow color at `360deg` (specifically, `#fa0`).
>
> The color of the gradient at any point is determined by first finding the unique ray anchored at the center of the gradient that passes through the given point. The point's color is then the color of the gradient line at the location where this ray intersects it.

Here is how the example from the specification renders:

Figure B.1: Rendering of `conic-gradient(red -50%, yellow 150%)`

Appendix C

List of commits

For historical purposes, this section provides a list of merged commits related to this project.

Bug ID	Commit message	Author	Reviewers	Merged on	Version
1614160	Add AngleOrPercentage to style system	ntim	emilio	2020-02-09	74
1614648	Make GradientItem and ColorStop support angular color stops	ntim	emilio	2020-02-13	75
1614890	Implement conic-gradient for WebRender graphics backend	ntim	nical, emilio, gw	2020-02-13	75
1615489	Refactor GenericGradient for conic-gradient support	ntim	emilio	2020-02-16	75
1615876	Minor gradient parsing cleanup	emilio	ntim	2020-02-16	75
1615614	Make brush_conic_gradient.glsl compile on GLES	ntim	gw	2020-02-18	75
1616106	Add more tests and reference images for conic-gradient wrench test suite	ntim	nical	2020-02-19	75
1614202	Implement parsing for CSS conic-gradient syntax	ntim	emilio	2020-02-19	75
1615862	Handle conic-gradients in nsCSSGradientRenderer for WebRender	ntim	mstange, emilio	2020-02-21	75
1616587	Implement conic-gradient for Skia graphics backend	ntim	lsalzman	2020-02-21	75
1617397	Update WPT expectations for conic-gradient	ntim	jgraham	2020-02-23	75
1619006	Fix max value in ReadElementConstrained call in RecordedEvent::ReadPatternData	ntim	nical	2020-03-02	75
1616255	Handle normalized conic-gradient color stop positions in WebRender	ntim	gw	2020-03-05	75
1620328	Set conic-gradient angle range on Skia	ntim	lsalzman	2020-03-09	75
1616986	Add WPT reftests for conic-gradient and stop normalization	ntim	emilio	2020-03-09	76
1620951	Fix definition of conic-gradient stopDelta in nsCSSGradientRenderer::Paint	ntim	emilio	2020-03-09	76
1621794	Add DevTools support for conic-gradient() and repeating-conic-gradient()	ntim	gl	2020-03-12	76
1618997	Improve serialization of conic and radial gradients	ntim	emilio	2020-03-28	76
1635818	Fix some conic gradient WPT accidentally passing when conic gradients are unsupported	ntim	emilio	2020-05-06	78

- The bug IDs can be consulted on Bugzilla [27], which contains all the information related to the commit.
- The author and reviewers columns both include the usernames for succinctness.
- The "Merged on" date is the date the commit reached the mozilla-central repository.

- The “Version” column mentions in which Firefox version the commit is available.

Appendix D

Support for mixing angle and percentages in `calc()` expressions

To test whether other browser engines (Blink and WebKit) support mixing angles and percentages in `calc()` expressions, it is possible to visit the following URL and check the rendering:

```
data:text/html,<div style="background-image: conic-gradient(red calc(10deg + 25%),
    gold); width: 100px;height:100px"/>
```

D.1 Blink

Blink was checked by using Google Chrome Canary version 84 (latest experimental version).

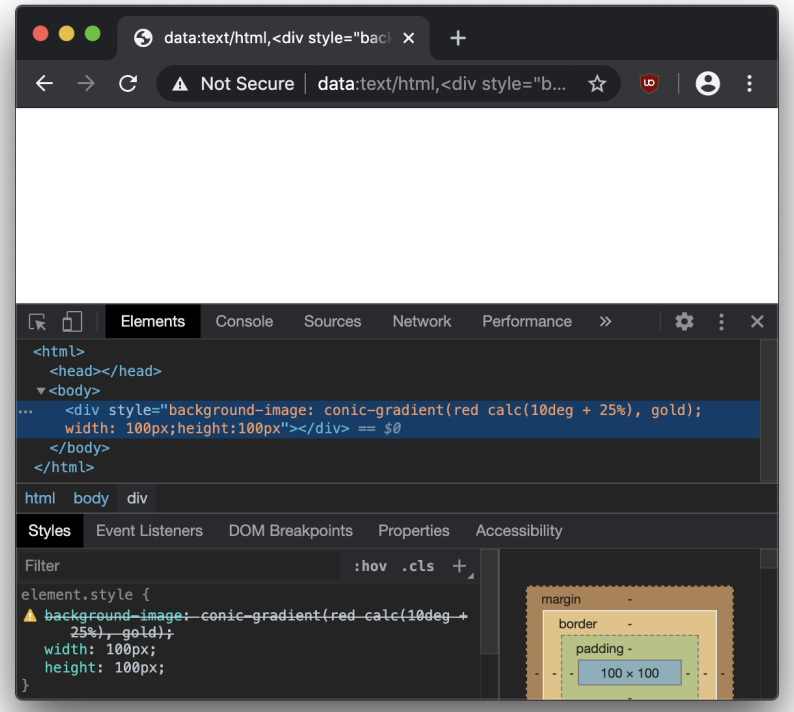


Figure D.1: Testcase on Chromium

The value is not recognised as valid by the developer tools and Chrome renders a blank page, meaning it is not supported.

D.2 WebKit

Safari 13.1 (latest stable version) was checked, for the WebKit rendering engine.

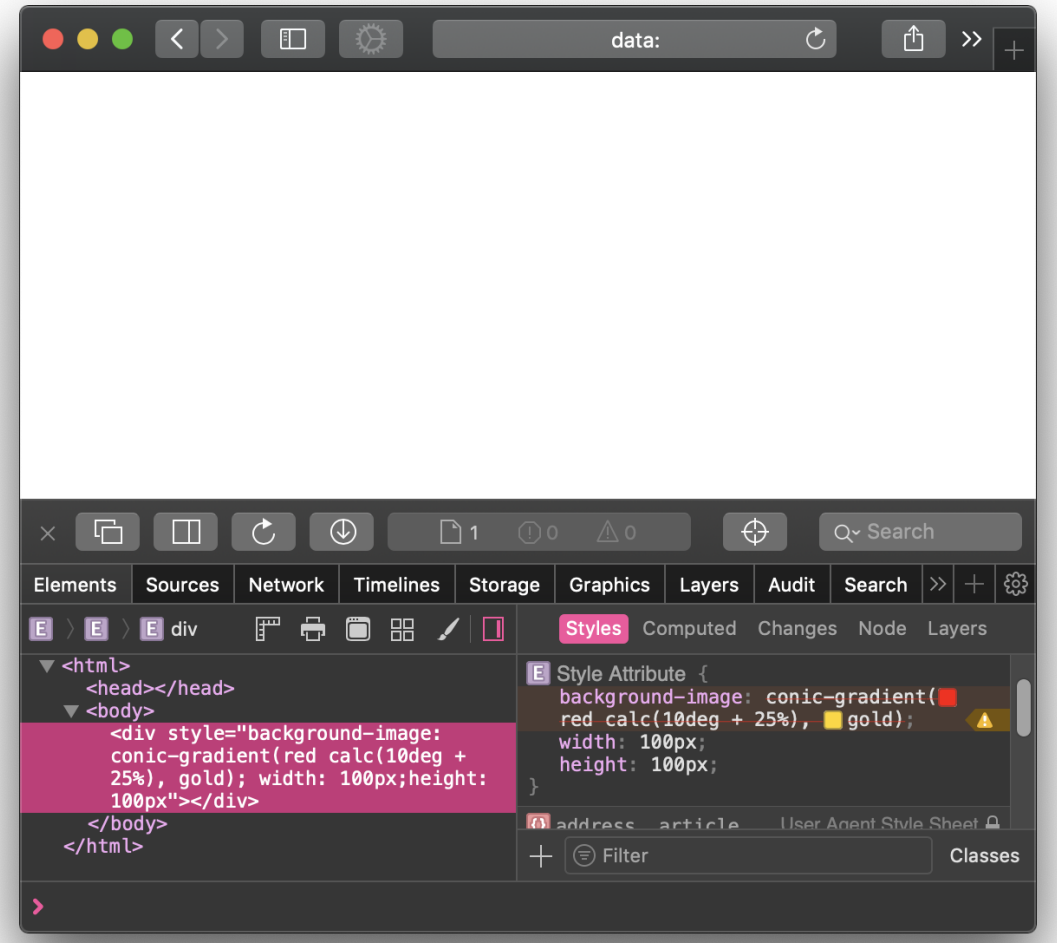


Figure D.2: Testcase on Safari

The results are similar to Chrome: the value is not recognised as valid and the page is rendered blank.

Appendix E

AngleOrPercentage code

E.1 Specified value implementation

```
/// <angle> | <percentage>
/// https://drafts.csswg.org/css-values/#typedef-angle-percentage
#[allow(missing_docs)]
#[derive(Clone, Copy, Debug, MallocSizeOf, PartialEq, SpecifiedValueInfo, ToCss, ToShmem)]
pub enum AngleOrPercentage {
    Percentage(Percentage),
    Angle(Angle),
}

impl AngleOrPercentage {
    fn parse_internal<'i, 't>(
        context: &ParserContext,
        input: &mut Parser<'i, 't>,
        allow_unitless_zero: AllowUnitlessZeroAngle,
    ) -> Result<Self, ParseError<'i>> {
        if let Ok(per) = input.try(|i| Percentage::parse(context, i)) {
            return Ok(AngleOrPercentage::Percentage(per));
        }

        Angle::parse_internal(context, input, allow_unitless_zero).map(AngleOrPercentage::Angle)
    }

    /// Allow unitless angles, used for conic-gradients as specified by the spec.
    /// https://drafts.csswg.org/css-images-4/#valdef-conic-gradient-angle
    pub fn parse_with_unitless<'i, 't>(
        context: &ParserContext,
        input: &mut Parser<'i, 't>,
    ) -> Result<Self, ParseError<'i>> {
        AngleOrPercentage::parse_internal(context, input, AllowUnitlessZeroAngle::Yes)
    }
}

impl Parse for AngleOrPercentage {
    fn parse<'i, 't>(
        context: &ParserContext,
        input: &mut Parser<'i, 't>,
    ) -> Result<Self, ParseError<'i>> {
        AngleOrPercentage::parse_internal(context, input, AllowUnitlessZeroAngle::No)
    }
}
```

Appendix F

Refactoring gradient code

The full definition of the structures before and after the refactor of the gradient code is available in this appendix. The full diff for the refactor can be found at [64].

F.1 Before

```
/// A CSS gradient.
/// <https://drafts.csswg.org/css-images/#gradients>
#[derive(Clone, Debug, MallocSizeOf, PartialEq, ToComputedValue, ToResolvedValue, ToShmem)]
#[repr(C)]
pub struct GenericGradient<
    LineDirection,
    LengthPercentage,
    NonNegativeLength,
    NonNegativeLengthPercentage,
    Position,
    Color,
> {
    /// Gradients can be linear or radial.
    pub kind: GenericGradientKind<
        LineDirection,
        NonNegativeLength,
        NonNegativeLengthPercentage,
        Position,
    >,
    /// The color stops and interpolation hints.
    pub items: crate::OwnedSlice<GenericGradientItem<Color, LengthPercentage>>,
    /// True if this is a repeating gradient.
    pub repeating: bool,
    /// Compatibility mode.
    pub compat_mode: GradientCompatMode,
}
pub use self::GenericGradient as Gradient;

/// A gradient kind.
#[derive(Clone, Copy, Debug, MallocSizeOf, PartialEq, ToComputedValue, ToResolvedValue, ToShmem)]
#[repr(C, u8)]
pub enum GenericGradientKind<
    LineDirection,
    NonNegativeLength,
    NonNegativeLengthPercentage,
    Position,
```

```
> {
    /// A linear gradient.
    Linear(LineDirection),
    /// A radial gradient.
    Radial(
        GenericEndingShape<NonNegativeLength, NonNegativeLengthPercentage>,
        Position,
    ),
}
pub use self::GenericGradientKind as GradientKind;
```

F.2 After

```
/// A CSS gradient.
/// <https://drafts.csswg.org/css-images/#gradients>
#[derive(Clone, Debug, MallocSizeOf, PartialEq, ToComputedValue, ToResolvedValue, ToShmem)]
#[repr(C)]
pub enum GenericGradient<
    LineDirection,
    LengthPercentage,
    NonNegativeLength,
    NonNegativeLengthPercentage,
    Position,
    Angle,
    AngleOrPercentage,
    Color,
> {
    /// A linear gradient.
    Linear {
        /// Line direction
        direction: LineDirection,
        /// The color stops and interpolation hints.
        items: crate::OwnedSlice<GenericGradientItem<Color, LengthPercentage>>,
        /// True if this is a repeating gradient.
        repeating: bool,
        /// Compatibility mode.
        compat_mode: GradientCompatMode,
    },
    /// A radial gradient.
    Radial {
        /// Shape of gradient
        shape: GenericEndingShape<NonNegativeLength, NonNegativeLengthPercentage>,
        /// Center of gradient
        position: Position,
        /// The color stops and interpolation hints.
        items: crate::OwnedSlice<GenericGradientItem<Color, LengthPercentage>>,
        /// True if this is a repeating gradient.
        repeating: bool,
        /// Compatibility mode.
        compat_mode: GradientCompatMode,
    },
}

pub use self::GenericGradient as Gradient;
```

Appendix G

nsCSSRenderingGradients changes

This is the combined diff of `nsCSSRenderingGradients.cpp` changes, over conic-gradient multiple commits (listed in chronological order):

1. Tim Nguyen - Bug 1615862 - Handle conic-gradients in nsCSSGradientRenderer for WebRender. r=emilio,mstange (`https://phabricator.services.mozilla.com/D65994`)
2. Tim Nguyen - Bug 1616587 - Implement conic-gradient for Skia graphics backend. r=lsalzman (`https://phabricator.services.mozilla.com/D63415`)
3. Tim Nguyen - Bug 1620328 - Set conic-gradient angle range on Skia. r=lsalzman (`https://phabricator.services.mozilla.com/D65910`)

```
@@ -225,6 +225,17 @@
   return MakeTuple(start, end, radiusX, radiusY);
 }

+// Compute the center and the start angle of the conic gradient.
+static Tuple<CSSPoint, float> ComputeConicGradientProperties(
+    const StyleGradient& aGradient, const CSSSize& aBoxSize) {
+  const auto& conic = aGradient.AsConic();
+  const Position& position = conic.position;
+  float angle = static_cast<float>(conic.angle.ToRadians());
+  CSSPoint center = ResolvePosition(position, aBoxSize);
+
+  return MakeTuple(center, angle);
+}
+
 static float Interpolate(float aF1, float aF2, float aFrac) {
   return aF1 + aFrac * (aF2 - aF1);
 }
@@ -459,7 +470,8 @@

 static ColorStop InterpolateColorStop(const ColorStop& aFirst,
                                       const ColorStop& aSecond,
-                                      double aPosition, const Color& aDefault) {
+                                      double aPosition,
+                                      const Color& aDefault) {
   MOZ_ASSERT(aFirst.mPosition <= aPosition);
   MOZ_ASSERT(aPosition <= aSecond.mPosition);

@@ -563,22 +575,40 @@
```

```
   return Some(pos.ResolveToCSSPixels(aLineLength) / aLineLength);
 }

-static nsTArray<ColorStop> ComputeColorStops(ComputedStyle* aComputedStyle,
-                                              const StyleGradient& aGradient,
-                                              CSSCoord aLineLength) {
-  auto items = aGradient.IsLinear() ? aGradient.AsLinear().items.AsSpan()
-                                    : aGradient.AsRadial().items.AsSpan();
+// aLineLength argument is unused for conic-gradients.
+static Maybe<double> GetSpecifiedGradientPosition(
+    const StyleGenericGradientItem<StyleColor, StyleAngleOrPercentage>& aItem,
+    CSSCoord aLineLength) {
+  if (aItem.IsSimpleColorStop()) {
+    return Nothing();
+  }
+
+  const StyleAngleOrPercentage& pos = aItem.IsComplexColorStop()
+                                          ? aItem.AsComplexColorStop().position
+                                          : aItem.AsInterpolationHint();

-  MOZ_ASSERT(items.Length() >= 2,
+  if (pos.IsPercentage()) {
+    return Some(pos.AsPercentage()._0);
+  }
+
+  return Some(pos.AsAngle().ToRadians() / (2 * M_PI));
+}
+
+template <typename T>
+static nsTArray<ColorStop> ComputeColorStopsForItems(
+    ComputedStyle* aComputedStyle,
+    Span<const StyleGenericGradientItem<StyleColor, T>> aItems,
+    CSSCoord aLineLength) {
+  MOZ_ASSERT(aItems.Length() >= 2,
              "The parser should reject gradients with less than two stops");

-  nsTArray<ColorStop> stops(items.Length());
+  nsTArray<ColorStop> stops(aItems.Length());

   // If there is a run of stops before stop i that did not have specified
   // positions, then this is the index of the first stop in that run.
   Maybe<size_t> firstUnsetPosition;
-  for (size_t i = 0; i < items.Length(); ++i) {
-    const auto& stop = items[i];
+  for (size_t i = 0; i < aItems.Length(); ++i) {
+    const auto& stop = aItems[i];
     double position;

     Maybe<double> specifiedPosition =
@@ -589,7 +619,7 @@
     } else if (i == 0) {
       // First stop defaults to position 0.0
       position = 0.0;
-    } else if (i == items.Length() - 1) {
+    } else if (i == aItems.Length() - 1) {
       // Last stop defaults to position 1.0
       position = 1.0;
     } else {
```

```
@@ -634,6 +664,21 @@
   return stops;
 }

+static nsTArray<ColorStop> ComputeColorStops(ComputedStyle* aComputedStyle,
+                                             const StyleGradient& aGradient,
+                                             CSSCoord aLineLength) {
+  if (aGradient.IsLinear()) {
+    return ComputeColorStopsForItems(
+        aComputedStyle, aGradient.AsLinear().items.AsSpan(), aLineLength);
+  }
+  if (aGradient.IsRadial()) {
+    return ComputeColorStopsForItems(
+        aComputedStyle, aGradient.AsRadial().items.AsSpan(), aLineLength);
+  }
+  return ComputeColorStopsForItems(
+      aComputedStyle, aGradient.AsConic().items.AsSpan(), aLineLength);
+}
+
 nsCSSGradientRenderer nsCSSGradientRenderer::Create(
     nsPresContext* aPresContext, ComputedStyle* aComputedStyle,
     const StyleGradient& aGradient, const nsSize& aIntrinsicSize) {
@@ -641,19 +686,26 @@

   // Compute "gradient line" start and end relative to the intrinsic size of
   // the gradient.
-  CSSPoint lineStart, lineEnd;
-  CSSCoord radiusX = 0, radiusY = 0;  // for radial gradients only
+  CSSPoint lineStart, lineEnd, center;  // center is for conic gradients only
+  CSSCoord radiusX = 0, radiusY = 0;    // for radial gradients only
+  float angle = 0.0;                    // for conic gradients only
   if (aGradient.IsLinear()) {
     Tie(lineStart, lineEnd) =
         ComputeLinearGradientLine(aPresContext, aGradient, srcSize);
-  } else {
+  } else if (aGradient.IsRadial()) {
     Tie(lineStart, lineEnd, radiusX, radiusY) =
         ComputeRadialGradientLine(aGradient, srcSize);
+  } else {
+    MOZ_ASSERT(aGradient.IsConic());
+    Tie(center, angle) = ComputeConicGradientProperties(aGradient, srcSize);
   }
   // Avoid sending Infs or Nans to downwind draw targets.
   if (!lineStart.IsFinite() || !lineEnd.IsFinite()) {
     lineStart = lineEnd = CSSPoint(0, 0);
   }
+  if (!center.IsFinite()) {
+    center = CSSPoint(0, 0);
+  }
   CSSCoord lineLength =
       NS_hypot(lineEnd.x - lineStart.x, lineEnd.y - lineStart.y);

@@ -677,6 +729,11 @@
   };
   renderer.mRadiusX = aPresContext->CSSPixelsToDevPixels(radiusX);
   renderer.mRadiusY = aPresContext->CSSPixelsToDevPixels(radiusY);
+  renderer.mCenter = {
+      aPresContext->CSSPixelsToDevPixels(center.x),
```

```
+      aPresContext->CSSPixelsToDevPixels(center.y),
+  };
+  renderer.mAngle = angle;
   return renderer;
 }

@@ -826,7 +883,8 @@
   double stopDelta = lastStop - firstStop;
   bool zeroRadius =
       mGradient->IsRadial() && (mRadiusX < 1e-6 || mRadiusY < 1e-6);
-  if (stopDelta < 1e-6 || lineLength < 1e-6 || zeroRadius) {
+  if (stopDelta < 1e-6 || (!mGradient->IsConic() && lineLength < 1e-6) ||
+      zeroRadius) {
     // Stops are all at the same place. Map all stops to 0.0.
     // For repeating radial gradients, or for any radial gradients with
     // a zero radius, we need to fill with the last stop color, so just set
@@ -869,8 +927,7 @@

     gradientPattern = new gfxPattern(gradientStart.x, gradientStart.y,
                                      gradientEnd.x, gradientEnd.y);
-  } else {
-    MOZ_ASSERT(mGradient->IsRadial());
+  } else if (mGradient->IsRadial()) {
     NS_ASSERTION(firstStop >= 0.0,
                  "Negative stops not allowed for radial gradients");

@@ -895,6 +952,9 @@
       matrix.PreScale(1.0, mRadiusX / mRadiusY);
       matrix.PreTranslate(-mLineStart);
     }
+  } else {
+    gradientPattern =
+        new gfxPattern(mCenter.x, mCenter.y, mAngle, stopOrigin, stopEnd);
   }
   // Use a pattern transform to take account of source and dest rects
   matrix.PreTranslate(gfxPoint(mPresContext->CSSPixelsToDevPixels(aSrc.x),
@@ -1120,7 +1180,8 @@
 void nsCSSGradientRenderer::BuildWebRenderParameters(
     float aOpacity, wr::ExtendMode& aMode, nsTArray<wr::GradientStop>& aStops,
     LayoutDevicePoint& aLineStart, LayoutDevicePoint& aLineEnd,
-    LayoutDeviceSize& aGradientRadius) {
+    LayoutDeviceSize& aGradientRadius, LayoutDevicePoint& aGradientCenter,
+    float& aGradientAngle) {
   aMode =
       mGradient->Repeating() ? wr::ExtendMode::Repeat : wr::ExtendMode::Clamp;

@@ -1136,6 +1197,8 @@
   aLineStart = LayoutDevicePoint(mLineStart.x, mLineStart.y);
   aLineEnd = LayoutDevicePoint(mLineEnd.x, mLineEnd.y);
   aGradientRadius = LayoutDeviceSize(mRadiusX, mRadiusY);
+  aGradientCenter = LayoutDevicePoint(mCenter.x, mCenter.y);
+  aGradientAngle = mAngle;
 }

 void nsCSSGradientRenderer::BuildWebRenderDisplayItems(
@@ -1151,8 +1214,10 @@
   LayoutDevicePoint lineStart;
   LayoutDevicePoint lineEnd;
```

```
   LayoutDeviceSize gradientRadius;
+  LayoutDevicePoint gradientCenter;
+  float gradientAngle;
   BuildWebRenderParameters(aOpacity, extendMode, stops, lineStart, lineEnd,
-                           gradientRadius);
+                           gradientRadius, gradientCenter, gradientAngle);

   nscoord appUnitsPerDevPixel = mPresContext->AppUnitsPerDevPixel();

@@ -1188,6 +1253,9 @@
   lineStart.x = (lineStart.x - srcTransform.x) * srcTransform.width;
   lineStart.y = (lineStart.y - srcTransform.y) * srcTransform.height;

+  gradientCenter.x = (gradientCenter.x - srcTransform.x) * srcTransform.width;
+  gradientCenter.y = (gradientCenter.y - srcTransform.y) * srcTransform.height;
+
   if (mGradient->IsLinear()) {
     lineEnd.x = (lineEnd.x - srcTransform.x) * srcTransform.width;
     lineEnd.y = (lineEnd.y - srcTransform.y) * srcTransform.height;
@@ -1199,8 +1267,7 @@
         mozilla::wr::ToLayoutPoint(lineEnd), stops, extendMode,
         mozilla::wr::ToLayoutSize(firstTileBounds.Size()),
         mozilla::wr::ToLayoutSize(tileSpacing));
-  } else {
-    MOZ_ASSERT(mGradient->IsRadial());
+  } else if (mGradient->IsRadial()) {
     gradientRadius.width *= srcTransform.width;
     gradientRadius.height *= srcTransform.height;

@@ -1211,7 +1278,15 @@
         mozilla::wr::ToLayoutSize(gradientRadius), stops, extendMode,
         mozilla::wr::ToLayoutSize(firstTileBounds.Size()),
         mozilla::wr::ToLayoutSize(tileSpacing));
+  } else {
+    MOZ_ASSERT(mGradient->IsConic());
+    aBuilder.PushConicGradient(
+        mozilla::wr::ToLayoutRect(gradientBounds),
+        mozilla::wr::ToLayoutRect(clipBounds), aIsBackfaceVisible,
+        mozilla::wr::ToLayoutPoint(gradientCenter), gradientAngle, stops,
+        extendMode, mozilla::wr::ToLayoutSize(firstTileBounds.Size()),
+        mozilla::wr::ToLayoutSize(tileSpacing));
   }
 }

 }  // namespace mozilla
```