The Gecko 3D transforms implementation is very similar to what was specified in the old TR - https://www.w3.org/TR/2013/WD-css-transforms-1-20131126

Transform-style property

- Transform-style's computed value is always the same as its specified value, and a value of 'preserve-3d' always creates a stacking context even if a flattening property is present.
- We also create a containing block for position:fixed (all) descendants if the value is 'preserve-3d', again, even if a flattening property is present. This was not mentioned in the spec, but is the case for 'transform'.
    - I considered transform-style:preserve-3d to effectively make the element transformed, so I think being a containing block makes sense.
- If there is also a flattening property (overflow != visible, opacity, filter etc), then we "require the user agent to create a flattened representation of the descendant elements before they can be applied, and therefore override the behavior of 'transform-style: preserve-3d':"

3D Rendering model

- Elements that have transform-style:preserve-3d and don't have a flattening property establish a 3d rendering context, or extend an existing one if they participate in it.
- Elements participate in a 3d rendering context if their parent (note: spec says containing block) establishes or extends a 3d rendering context.
- Content rendered for elements that establish/extend a 3d rendering context is included in the rendering (as well as elements that just participate).
- Since transform-style:preserve-3d is what establishes the 3d rendering context, siblings with preserve-3d under a transform-style:flat (default styled) parent are considered independent contexts that follow normal CSS z-index sorting rules, not 3d sorting.
- The post-transform rendering of the element that creates the 3d rendering context is what gets flattened to 2D and drawn into the outer stacking context.
- Planes (underdefined, see below for explanation) within a 3d rendering context are sorted by depth.
- 3d rendering contexts (and elements with a normal 3d transform) don't sort amongst each other.

Explanation of how we generate planes, and handle coplanarity:

```
        <div style="background-color:red"></div>
        <div style="transform: B; background-color:green"></div>
        <div style="background-color:orange"></div>
        <div style="background-color:black; position:relative; z-index:2"></div>
</div>
```

We follow the normal CSS2.1 z-index rules for painting a stacking context, and wrap consecutive non-transformed items into a single plane.

```
plane(transform: A, preserve-3d)
        plane(transform: identity)
                background-color:blue
                background-color:red
                background-color:orange
        plane(transform: B)
                Background-color:green
        plane(transform: identity)
                Background-color:black
```

During compositing we flatten preserve-3d containers transform into their children, so that the 3d is preserved, but keep the rendering context together to be sorted.

```
Sorting group
        plane(transform: A)
                background-color:blue
                background-color:red
                background-color:orange
        plane(transform: A*B)
                Background-color:green
        plane(transform: A)
                Background-color:black
```

This scene is then sorted and composited.

Existing work to explain this:
https://docs.google.com/document/d/1FIQW9qVPbZxn0pifFOXWWK0-7fXrjlSeYeZN7wHmlHo/edit#heading=h.jd53mnjhsgjb

Backface-visibility handling

Gecko works hard to handle this property so that it applies only to the element itself, and not descendants (as specified). It would be better to not.

The code for combining consecutive non-transformed items into a single plane also takes backface-visibility into account and splits a new plane at each transition of the computed value of backface-visibility.

The visibility of a backface is determined using the transform of the plane up to the root of the 3d rendering context. If there are nested 3d rendering contexts (intentionally, or due to a flattening property), or 3d transforms, then this may not be the same as whether the backface is visible to the user.

Perspective handling

Current spec wording is that we should look to the containing block (which gecko implements), but initial tests (and code reading) suggest that blink/WebKit are using the parent 'layer'. Stacking context might be closer to this (but not identical).

Z-index creates a stacking context, but probably not a layer in WebKit/blink. Should that intercept perspective lookup? It currently doesn't.

http://webkit.crisal.io/webkit/rev/6cc00baaf9835dc4abdf7b16e72cff60ecd7bb88/Source/WebCore/rendering/RenderLayer.cpp#4955

https://bugzilla.mozilla.org/show_bug.cgi?id=1536378
https://bugs.chromium.org/p/chromium/issues/detail?id=943503
https://webcompat.com/issues/27090

Tests

Transform-style: preserve-3d creates a stacking context, even when a flattening property is applied. Blink does this, WebKit doesn't. Also shows that the computed value of transform-style doesn't change when flattened (true in gecko and blink, WebKit changes computed value to flat).

Transform-style: preserve-3d creates a containing block, even when a flattening property is applied. Blink appears to create a CB only when not-flattened, WebKit doesn't seem to ever.

Basic 3d cube. Appears to work the same in all.

Two cubes don't intersect since they establish independent rendering contexts. Gecko and blink seem to agree here, WebKit does intersection between the cubes.

Two cubes intersect when the common parent 'scene' has preserve-3d. All appear to work the same here.

Two cubes don't intersect when the scene has preserve-3d, but has a flattening property too. Gecko and blink agree, WebKit continues to sort.

Elements use their direct parent to determine if they extend an existing rendering context. Only Gecko is doing this, blink and WebKit are still preserving 3d through an empty .

Flattening properties cause flattening, even if they're neither a stacking context nor a containing block. Irrelevant in gecko (since we need preserve-3d to attempt to be a 3d rendering context that can be flattened, and that makes us a SC/CB), true in both WebKit/blink. Does it create a SC/CB in this case? Another example demonstrating that it does not in Blink.

Nested transforms without preserves-3d. What is the flattening matrix behavior?

Flattening properties don't create a 3d rendering context in WebKit. The second example shows two preserve-3d planes within a div with a flattening property (but nothing else, so it's not a CB or SC). It looks like this doesn't create a 3d rendering context, and planes are just sorted normally. Sorting also seems to depend on scroll position?
As above, Gecko and blink never sort preserve-3d siblings that aren't within an outer transform-style:preserve-3d.

Perspective creates a stacking context. Works in all.

Perspective creates a containing block. Works in Gecko and blink, doesn't in WebKit.

Perspective ignores an element that isn't a containing block between transformed and perspective element. The spacer isn't the containing block for the transformed element, so this should be ignored. Works in all.

Perspective doesn't ignore a containing block between transformed and perspective element. The spacer is the containing block for the transformed element, so we should be looking at it for the perspective value. Spacer is also a pseduo-stacking context. Works in Gecko (no perspective), WebKit/blink still apply the perspective.

Perspective ignores an opacity stacking context between transformed and perspective element. spacer is a stacking context, but not a containing block, should be ignored. Works in Gecko (has perspective), WebKit/blink remove the perspective.

Perspective ignores a z-index stacking context between transformed and perspective element. Same as above, except using z-index to create the stacking context. Now works in all (perspective always applied). Suggests that 'stacking context' is insufficient to describe the existing behaviour.

Backface visiblity tests:

Testcase for backface visibility on a normal transformed element. Works in all.

Testcase for backface visibility on a transformed child of preserve-3d. Works in all.

Testcase for backface visibility on an untransformed child of preserve-3d. Works in all.

Testcase for backface visibility of children of an untransformed element with backface-visiblity:hidden.
Webkit and blink hide the backface of 'card front' and descendants (which includes the contents of 'inner') as a (psuedo-)stacking context. Gecko doesn't since it only hides the backface of the 'card front' element itself, and not that of descendants.

Testcase for backface visibility of positioned descendants of an untransformed element with backface-visiblity:hidden.
Inner is visible in all. In Gecko because it's per-element and isn't considered for 'inner'. WebKit/blink looks to be because backface-visibility:hidden creates a pseudo-stacking context (CSS2 E.2.5 [1]), and the position:fixed child is not included in that.

Testcase for backface visibility of stacking-context descendants of an untransformed element with backface-visibility:hidden.
Same as above, except the 'inner' descendant now has a stacking-context property (opacity), instead of being positioned fixed.

Testcase for backface visibility of positioned children with backface-visibility of an untransformed element with backface-visiblity:hidden.
Same as the previous test, except now 'inner' has the backface-visibility:hidden property. We'd expect this not to have any effect, as inner doesn't participate in a 3D rendering context (it's not a child of an element with transform-style:preserve-3d), but it does have an effect in blink/WebKit, and hides inner.
I think this is due to 'card front' being a pseudo-stacking context, which doesn't capture 'inner', and means that these two Elements are effectively siblings from the perspective of painting/compositing layers. 'Inner' also considers 'outer' to be its parent due to this, which is why it takes backface-visibility into account.
If we want to spec this behaviour, then I think the description of 3D rendering contexts will need a way to define that 'inner' is participating in the 3D rendering context created by 'outer'. Positioned elements are included if their containing block are preserve-3d, normal elements if their parent is?

---

[1] "treat the element as if it created a new stacking context, but any positioned descendants and descendants which actually create a new stacking context should be considered part of the parent stacking context, not this new one." - https://www.w3.org/TR/CSS2/zindex.html#painting-order

Backface visibility is computed relative to root of 3d rendering context (and not the screen/viewer). Works in all.

Try to find ways to show our plane behaviour.