Plan for compositing triggers in SPv2

Proposal
- Algorithm step 2
- Algorithm step 4

Background on SPv1
- Compositing triggers
- Behavior of transform, effects and clips during Blink paint

Proposal

1. **Direct reasons on paint property nodes**: Compute direct compositing reasons related to property tree nodes (e.g. 3D transform) in PaintPropertyTreeBuilder. Store them on the transform or effect node that corresponds to the direct reason. (*not* on PaintChunks). Direct reasons not related to property tree nodes (e.g. “will-change: transform”) are not computed here.
2. Compute final composited layers via a **small variation on the algorithm** (see below) towards the end of this document. The difference is that the iteration is over PaintChunks, and “CanMergeInto” is implemented by this algorithm:
    a. ~~**Direct reasons on PaintChunks**: If the PaintChunk has a direct reason for compositing (e.g. “will-change: transform”) then it gets a new layer.~~ (we decided that these are not necessary, and all reasons known should probably be on property tree nodes)
    b. **Merging into an existing composited layer / squashing**: Failing (a), walk up the property tree state of the PaintChunk, looking for one of the following. Note that this means the ancestor for merging must have property tree nodes which are ancestors of each of the property tree nodes of the PaintChunk.
        i. the nearest existing composited layer that has the same state as an ancestor property tree state. If found, return true.
        ii. or if a node is found which has a direct compositing reason, stopping and allocating a new composited layer. If found, return false.
3. **Overlap test**: If a composited layer was found for which CanMergeInto returned true, “Overlaps” is implemented by overlap testing against any composited layers which were created after that candidate merge layer.
4. **Optimizing away non-composited paint property nodes**: When merging or creating a new layer due to 2(i) or (3), the property tree nodes traversed in 2(b) will be collapsed into the PaintChunk before merging or creating a new composited layer. See below for pseudocode.

The above algorithm has been prototyped in this CL. The output of the compositing/layer-creation/examples.html layout test in SPv1 mode and SPv2 mode are below. Note that the number of layers and their content is the same.

Algorithm step 2

.. is based on, from the other document:

Input: vector<SimpleLayer*> simple_layer_list
Output: vector<Layer*> layer_list

```
Algorithm layerize(simple_layer_list):

vector<Layer*> layer_list
for (int i = 0; i < simple_layer_list.size(); ++i) {
  for (int j = layer_list.size() - 1; j >= 0; --j) {
    if (simple_layer_list[i]->CanMergeInto(layer_list[j])) {
      layer_list[j]->AddSimpleLayer(simple_layer_list[i]);
      break;
    }
    if (simple_layer_list[i]->Overlaps(layer_list[j]) || j == 0) {
      Layer* new_layer = new Layer();
      new_layer->AddSimpleLayer(simple_layer_list[i]);
      layer_list.push(new_layer);
      break;
    }
  }
}
```

Algorithm step 4

... pseudocode is as follows:

Note that it is a precondition that `ancestorPropertyTreeState` must an ancestor of `childPropertyTreeState`. (step 2b ensures this.)

Define a total ordering <= on property tree nodes such that node a <= node b if a's clip, effect or transform applies before b, or it's equal to that node.

```
Algorithm flatten(childPropertyTreeState, ancestorPropertyTreeState)

currentTransformSpace = childPropertyTreeState.transform
```

```
Iterate over the property tree nodes in backwards total ordering, starting at
ancestorPropertyTreeState and ending at childPropertyTreeState:
      If the next node is a clip node:
            LowestClipNode = current clip node
      If the next node is an effect node:
            map its bounds and clip up into currentTransformSpace (*),
            And record a paired display item for it in that space.
      If the next node is a transform node:
            Collect all clips between lowestClipNode and the highest clip
            Node in currentTransformSpace.
            Map those clips to currentTransformSpace into a combined clip.
            Record a paired display item for that clip in
            currentTransformSpace.

            Record a paired display item for the transform.

            currentTransformSpace = current transform

(*) An effect can have a compositing effect like opacity, and also a filter.
Both have bounds and clip rects. These need are the items that need to be
mapped up into currentTransformSpace
```

Background on SPv1

Compositing triggers

A run-down of compositing trigger bits in SPv1 are here.

SPv1 computes and applies compositing triggers via two (and a bit) classes:

CompositingReasonFinder: computes “direct” and “potential” compositing reasons for a LayoutObject. Doesn’t need any tree walks. “Direct” reasons are those which require compositing. “Potential” reasons are those which might cause compositing, if certain circumstances apply. There are also some global options specifed on it like LCD text.

CompositingRequirementsUpdater: a tree walk over PaintLayers that applies the reasons found via CompositingReasonFinder to allocate composited layers for some of them. A PaintLayer can get composited if:

1. It has a direct compositing reason
2. It has a potentially direct compositing reason and the necessary conditions apply to upgrade that potential reason to a direct reason. These reasons all have to do with whether ancestors or descendants composite.

    a. Example for ancestor: scrolling will composite always (regardless of LCD) if an ancestor scroller is also composited.
    b. Example for descendant: effects like opacity, blending, and filters only trigger compositing if there have composited descendants or accompanying direct reasons on the same object.
3. It overlaps a PaintLayer that paints earlier but into a different backing that is not already behind the “current” one. The current one is usually the backing inherited from the nearest compositing ancestor, though there are a few complicated rules about negative z-index children, etc.

CompositingLayerAssigner: two adjacent PaintLayers in paint order which both received compositing due to overlap (item 3 above) will get squashed down into one layer if they have compatible scrolling, transform, and effect states, and the squashing layer won’t get too sparse.

The final compositing reasons are recorded in each PaintLayer as PaintLayer::compositingReasons(), though they are not really read later on except for debugging.

Behavior of transform, effects and clips during Blink paint

In SPv1, many 2D transforms, clips and effects are painted within a single GraphicsLayer (cc::Layer). Painting recurses through each PaintLayer that paints into the same GraphicsLayer backing. Transforms and effects apply to entire subtrees of PaintLayers, but clips may not due to containing block semantics.

Definitions:

- The *2D transform ancestor* of a PaintLayer is the containing PaintLayer which paints into the same backing but has a 2D transform.
- The clips applied by a PaintLayer are overflow, contains-paint, CSS or SVG root viewport clips it applies.

When painting contents of a PaintLayer, it proceeds like this:

If the PaintLayer has a transform:

1. Apply all inherited clips that apply to this PaintLayer but are below the containing 2D transform, if any.
2. Apply any transform, if the PaintLayer has one.
3. Apply any effects, if the PaintLayer has them.
4. Apply any clip this PaintLayer applies locally, if needed depending on the paint phase
5. Recurse into child PaintLayers (may be before step 4 for negative z-index, etc.)

If the PaintLayer has a transform:

1. Apply any effects (same as step 3 above)

2. Apply all inherited clips (same as step 1 above)
3. Apply any clip this PaintLayer applies locally, if needed depending on the paint phase (same as step 4 above). (Merge this with #2 also, I think?)
4. Recurse.

Observations:

- Multiple clips are applied at once in step 1
- Clips are always applied in the same transform space as the clip
- Clips are pushed down inside effects (e.g. a clip from step 1 may be above an effect from an ancestor PaintLayer). [This is likely not correct for pixel-moving filters]