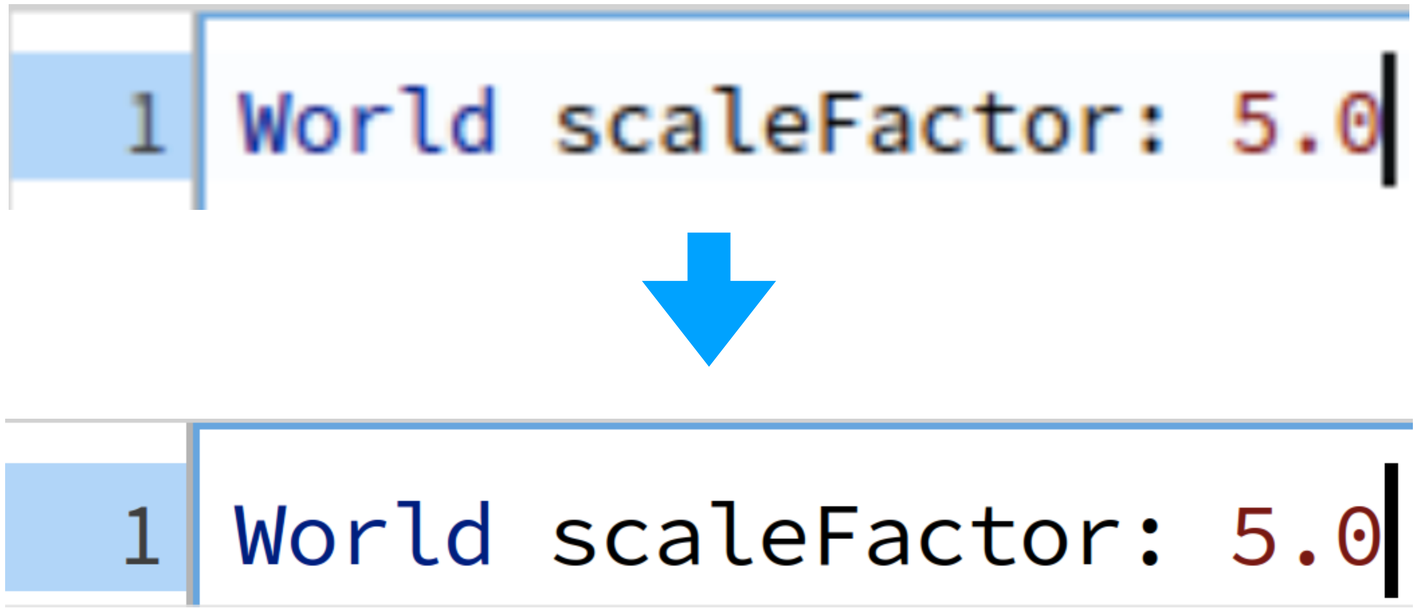
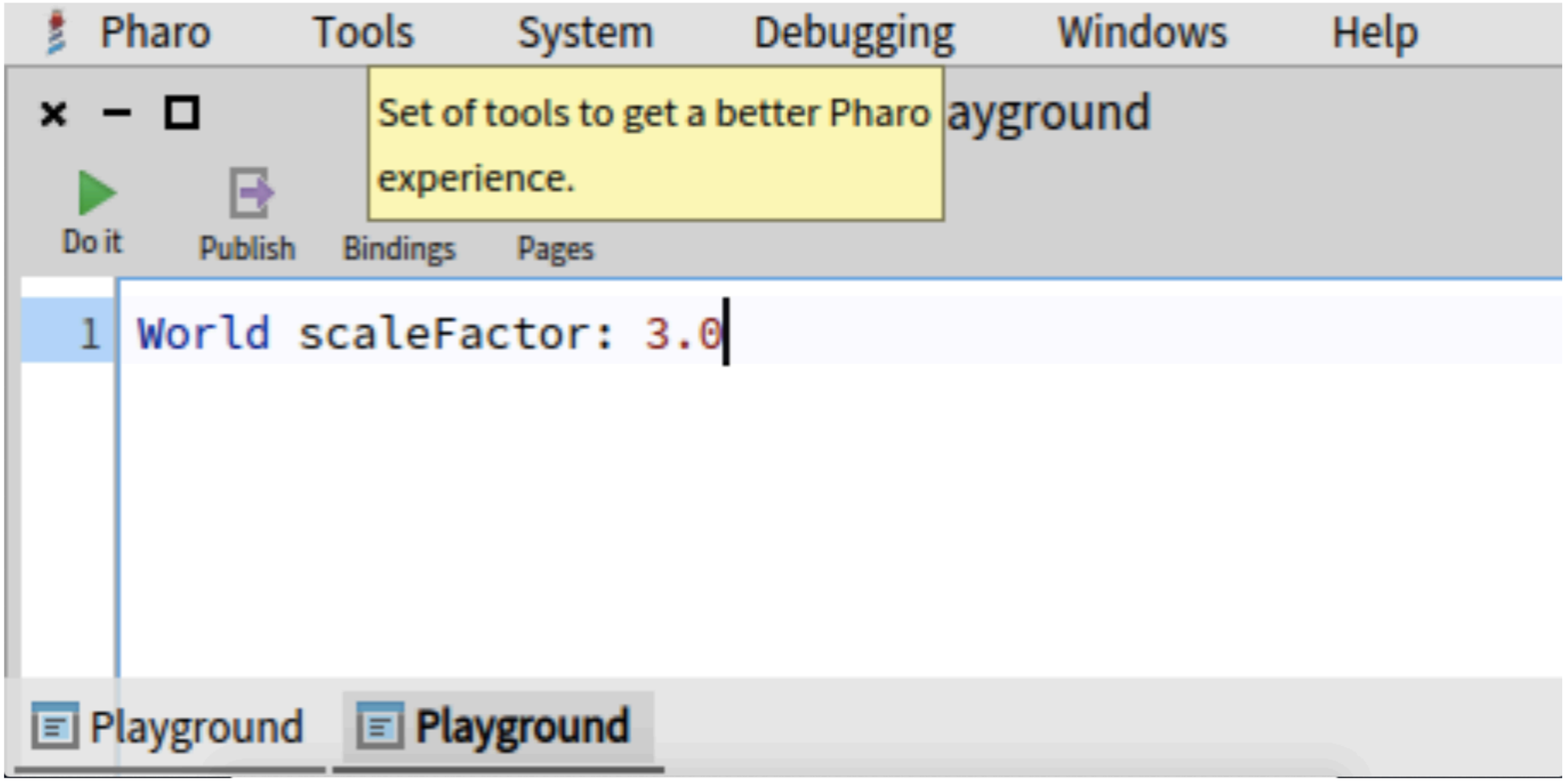
Form Scene Graph
A New UI Rendering Framework

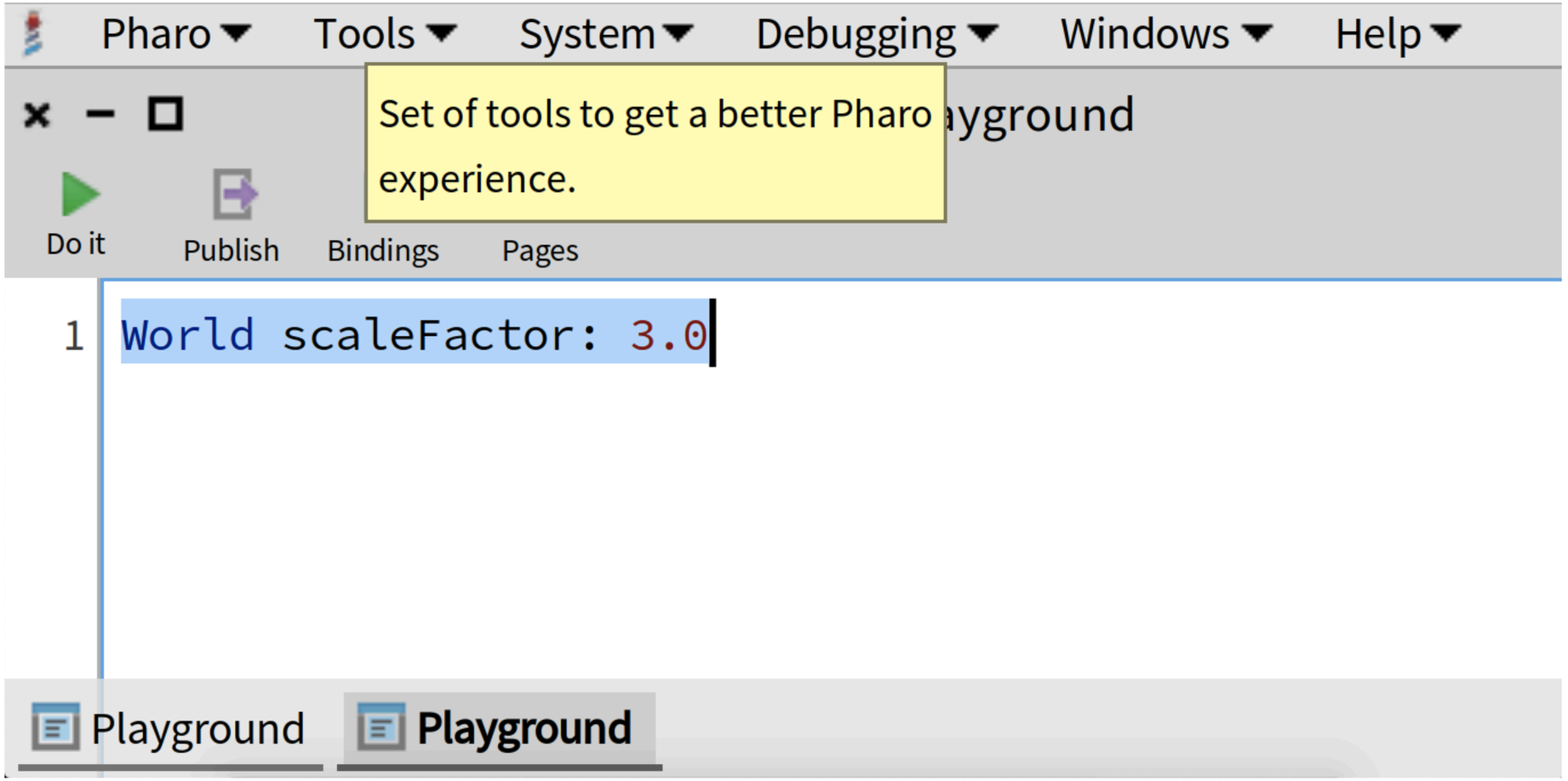
Ronie Salgado
RMoD, Inria, Lille, France



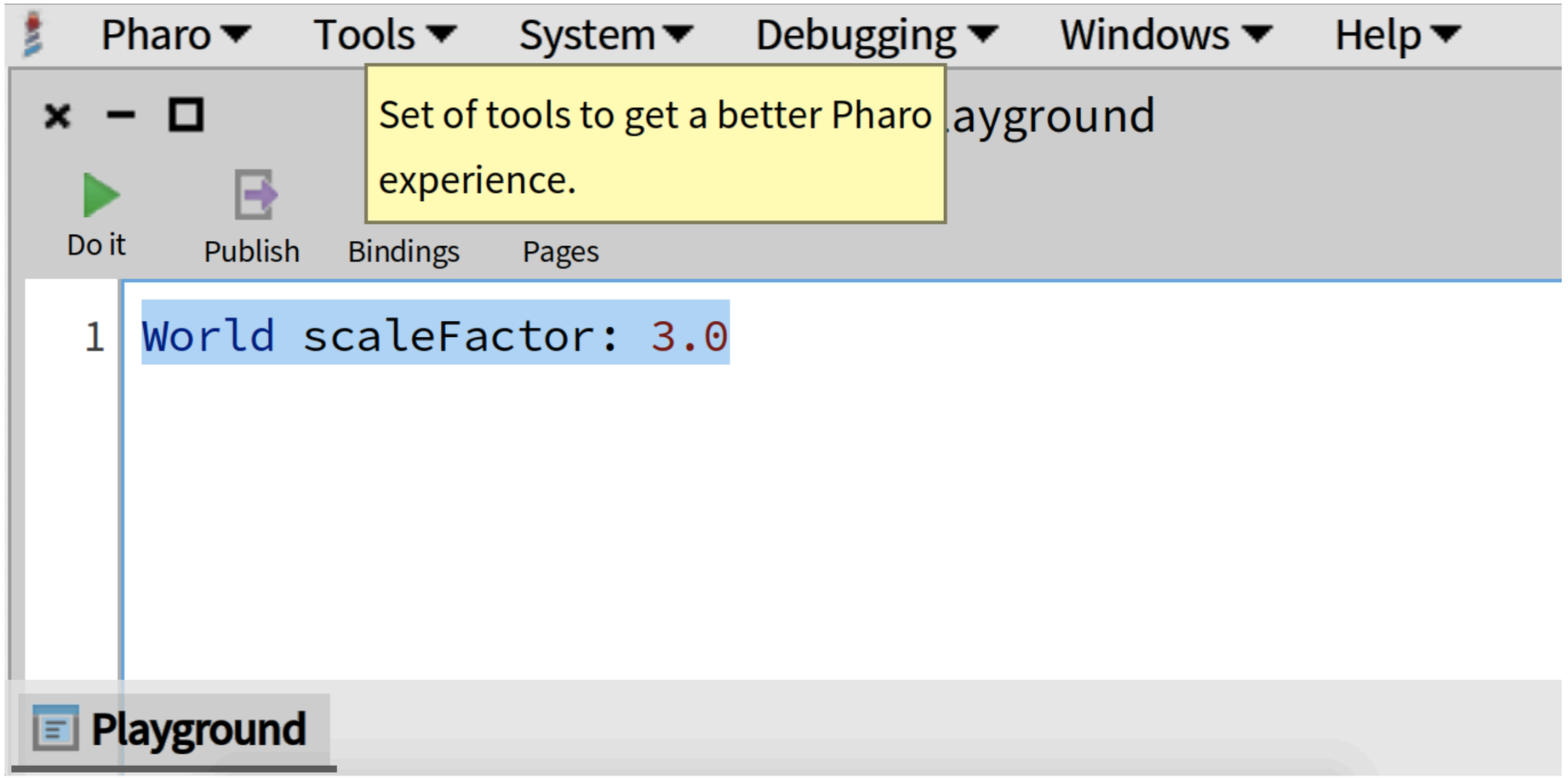
UI Scaling
Current Method - #fullDrawOn: & Canvas



UI Scaling
Scene Graph - Athens Backend



UI Scaling
Scene Graph - SDL2 Backend



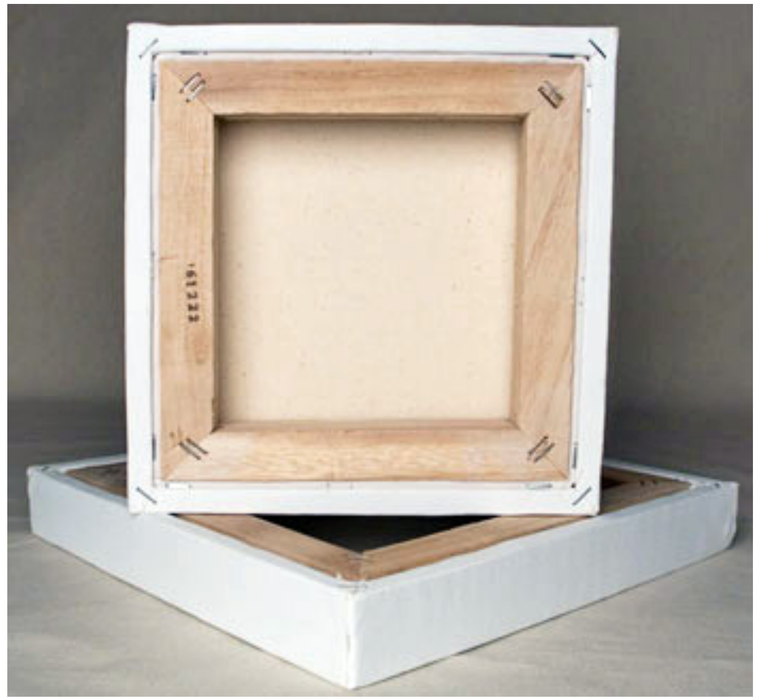
How to Install

GitHub: https://github.com/ronsaldo/form-scene-graph

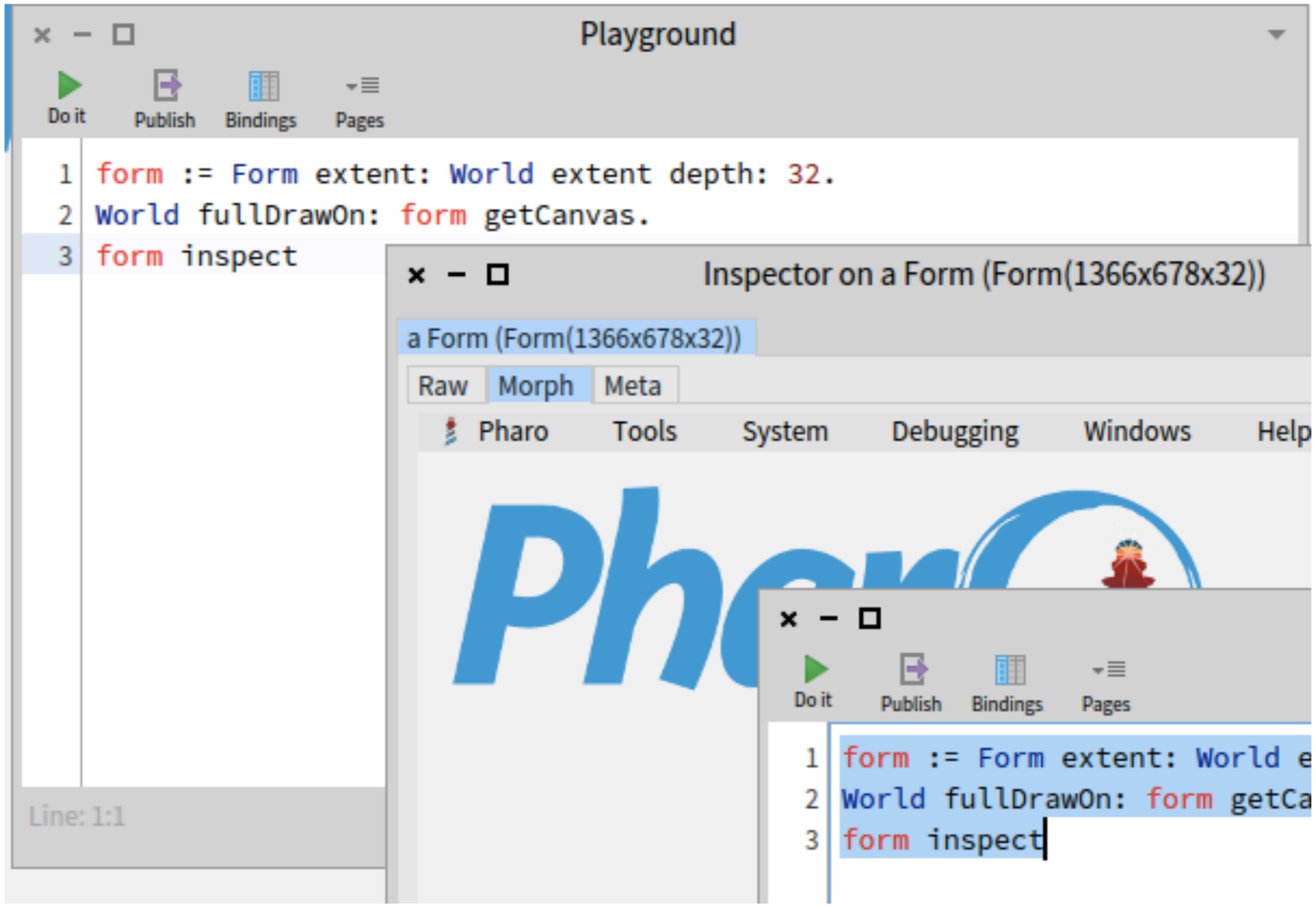
```
Metacello new
  baseline: 'FormSceneGraph';
  repository: 'github://ronsaldo/form-scene-graph';
  load.
```

The Current Method: Canvas Metaphor

- Start from Morph >> #fullDrawOn: aCanvas
- Pre-order traversal of Morphs.
- Sequential and immediate drawing on a Form Canvas.
- Requires immediate pixel access
  - Difficult to accelerate with GPU.
- Large API: Canvas selectors size => 89
- Non-scalable with arbitrary affine transforms.



How to Draw Manually the world



Morph Drawing Sequence

- fullDrawOn: aCanvas -> on: Error do: [.. drawRedScreenOfDeath ..]

    1. drawOn: aCanvas -> Draws the content of the morph itself

    2. drawSubmorphsOn: aCanvas -> Draw the children morphs on top.

    3. Additional effects (drawDropHighlightOn: and drawMouseDownHighlightOn:)

- In parallel we have fullDrawOnAthensCanvas:

- If we want more rendering mechanisms with this approach, all Morphs and subclasses must be extended, per-backend.

What Other UI Frameworks are Moving To:

- Gtk 3 uses Cairo (Athens) for rendering. They complaint about its performance.
- Gtk 4 is introducing the Gtk Scene Graph (OpenGL and Vulkan).
- QT already uses a scene graph.
- 3D rendering frameworks are based on scene graphs.
- Spec 2 has a Gtk backend -> We need to integrate Morphic on Gtk.

Taking Inspiration on GSK

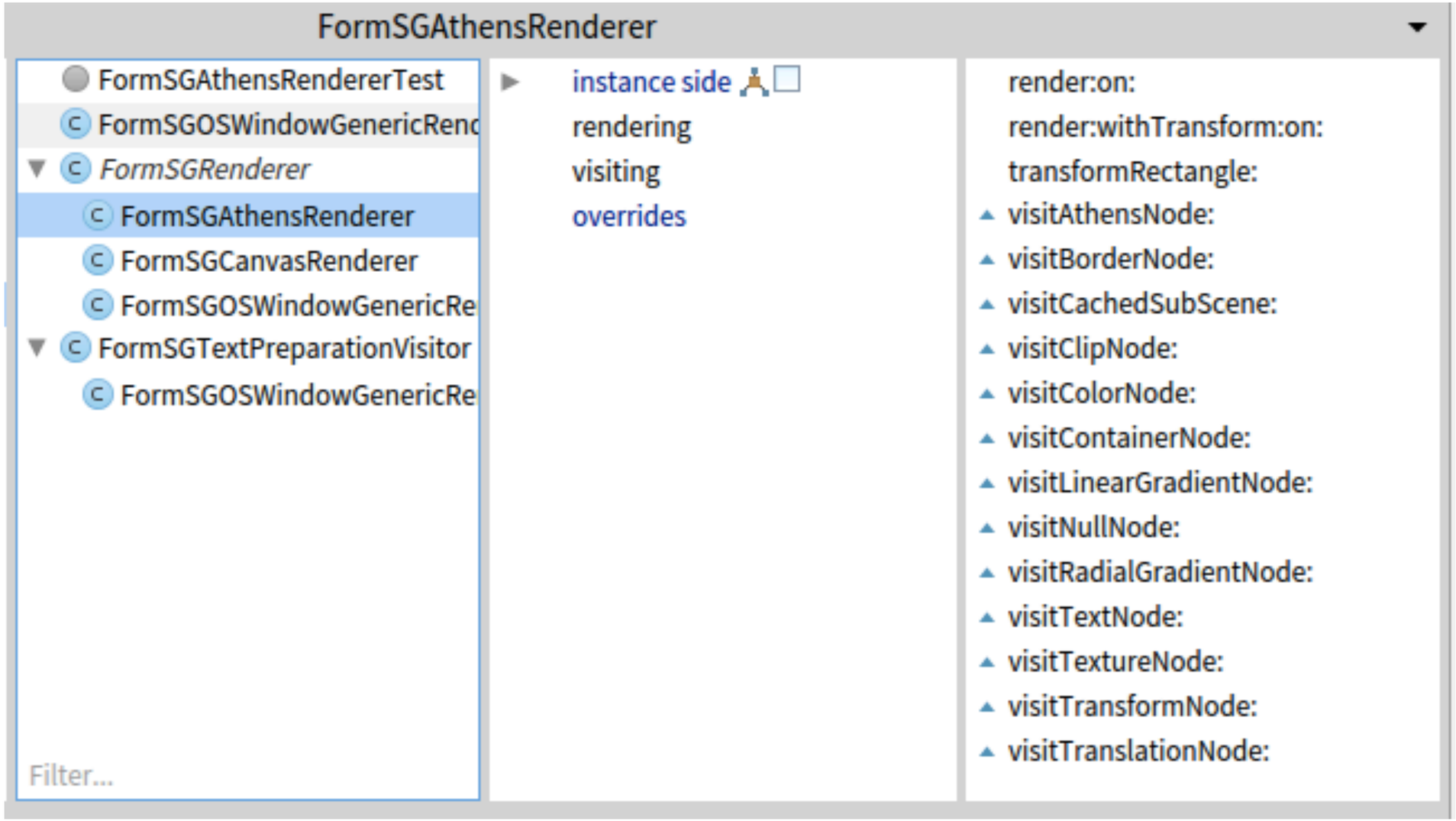
- A rendering tree.
- All shapes are rectangles.
- They are easy to draw individually.
- These shapes are composed.
- Selection of nodes based on CSS styling.
- Complex shapes are drawn statically with Cairo into textures. They are composed as rectangles.

```
typedef enum {
  GSK_NOT_A_RENDER_NODE = 0,
  GSK_CONTAINER_NODE,
  GSK_CAIRO_NODE,
  GSK_COLOR_NODE,
  GSK_LINEAR_GRADIENT_NODE,
  GSK_REPEATING_LINEAR_GRADIENT_NODE,
  GSK_BORDER_NODE,
  GSK_TEXTURE_NODE,
  GSK_INSET_SHADOW_NODE,
  GSK_OUTSET_SHADOW_NODE,
  GSK_TRANSFORM_NODE,
  GSK_OPACITY_NODE,
  GSK_COLOR_MATRIX_NODE,
  GSK_REPEAT_NODE,
  GSK_CLIP_NODE,
  GSK_ROUNDED_CLIP_NODE,
  GSK_SHADOW_NODE,
  GSK_BLEND_NODE,
  GSK_CROSS_FADE_NODE,
  GSK_TEXT_NODE,
  GSK_BLUR_NODE,
  GSK_OFFSET_NODE,
  GSK_DEBUG_NODE
} GskRenderNodeType;
```

Form Scene Graph Design

- Copy the nodes from GSK in Pharo classes.
- Builder class for instancing nodes. Some Canvas APIs are emulated here, but they produce these same nodes.
- Renderer backends are implemented as visitors.
- Few nodes are needed for drawing most of Morphic.

The Size of a Renderer Backend



Rendering Morphic with the Scene Graph

- Like in the previous case, Morphic has to be extended:
- #buildFullSceneGraphWith:
  - #buildSceneGraphNodeWith: (Most morphs need to override this)
  - #buildClippedChildrenSceneGraphNodeWith:
- Once a Morph is adapted for building a scene graph, it can be rendered with all of the backends.
- The scene graph by definition is scalable (Hi-DPI support).

Some Morph Adaptations

Morph >>

```
buildSceneGraphNodeWith: builder
    ^ builder fillRectangle: self bounds fillStyle: self fillStyle borderStyle: self borderStyle
```

FormSGBuilder - Extension for Morphic:

```
fillRectangle: aRectangle fillStyle: aFillStyle borderStyle: aBorderStyle
    (aBorderStyle isNil or: [aBorderStyle width < 0]) ifTrue: [
        ^ self fillRectangle: aRectangle style: aFillStyle
    ].

    ^ (self fillRectangle: (aRectangle insetBy: aBorderStyle width) style: aFillStyle) ,
        (self borderRectangle: aRectangle style: aBorderStyle)
```

ImageMorph >>

```
buildSceneGraphNodeWith: builder
    ^ (builder textureForm: image at: self innerBounds origin opaque: self isOpaque) ,
        (builder borderRectangle: self bounds style: self borderStyle).
```

Performance

- *AbstractWorldRenderer startProfilingRenderingTime* enables displaying the rendering time in the top-level corner.

- **Athens** Backend has CPU rendering. It is comparable to the previous renderer. Performance degrades with resolution increments. (26 - 80 ms)

- **OSWindowGenericRenderer** Backend (**SDL2**) may use the GPU. Has better performance in general, but gradient nodes are difficult to implement. (3 and 40 ms)

- **OpenGL ES** backend is faster and better than the OSWindowGenericRenderer. Requires compiling and distributing **ANGLE** for OS X and Windows.

- **AbstractGPU** backend (**Vulkan**, **Metal**, **Direct3D12**) slightly faster than the OpenGL ES backend. It suffers of stability issues (crashes on window resize).

Text Rendering

- This is a difficult process.
- Text rendering is the bottleneck.
- Subpixel antialiased font rendering ~~ Alpha Blending!
  - Each color channel is used as a separate alpha factor!
  - Two Pass Algorithm:
    1. Subtract the color mask from the background:

       glBlendFunc(GL_ZERO, GL_ONE_MINUS_SRC_COLOR)

    2. Add the color mask multiplied by the text color:

       glBlendFunc(GL_SRC_COLOR, GL_ONE)

  - One Pass rendering is possible with shaders and dual source blending.

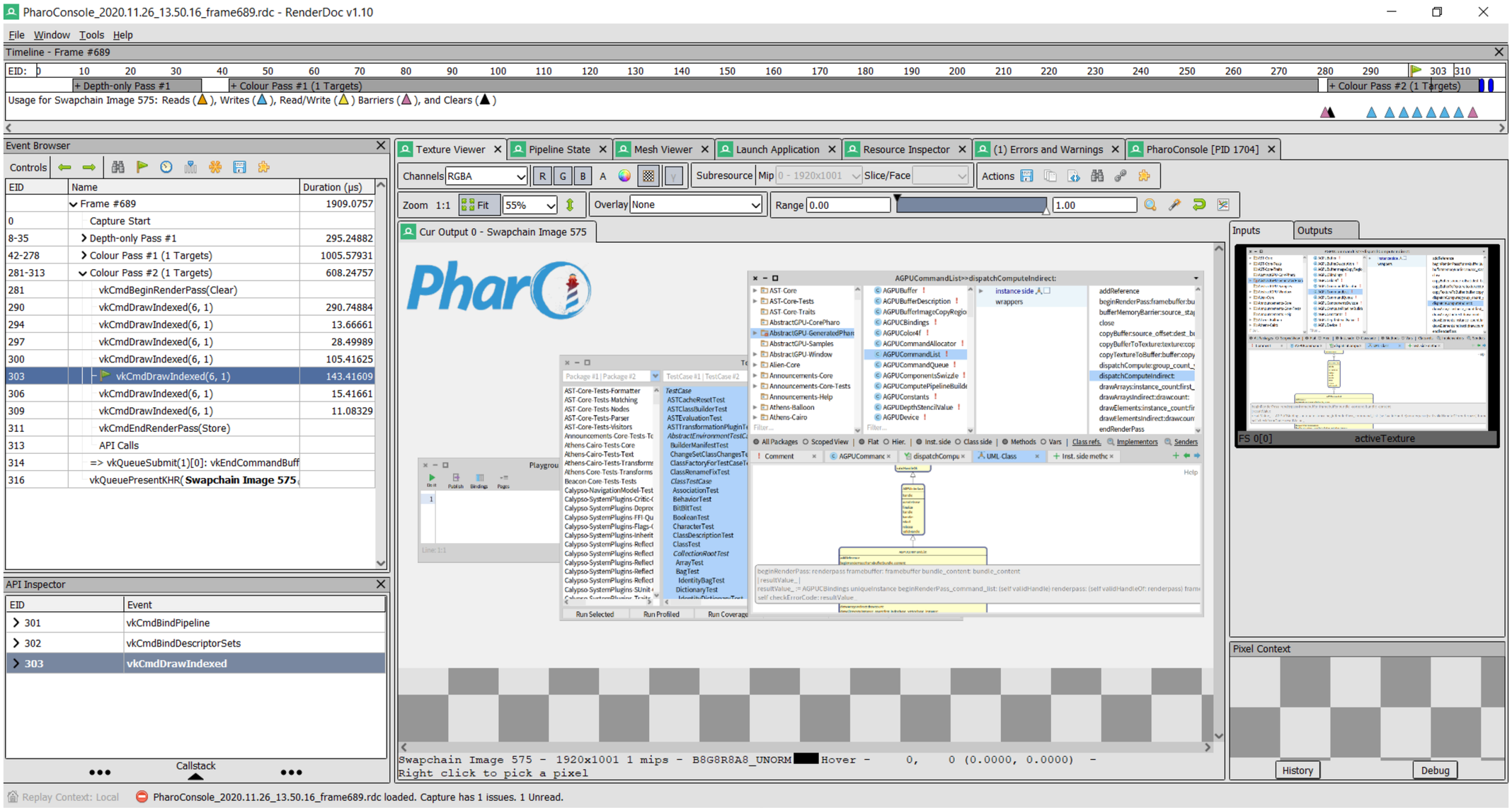
Scaling Text Rendering

- In the Athens backend, we just leave Cairo perform this task.
- Each renderer has a surfaceScaleFactor property.
- The active transform is not used for computing the text font scale to prevent thrashing the glyph cache on a dynamic scale change.
- scaledFont := font withSizeIncrementedBy: (font pointSize*surfaceScaleFactor) - font pointSize
- If the current transform scale = surfaceScaleFactor, and there is no rotation, then attempt pixel perfect blitting to avoid blurring:
  - Compute the destination pixel and blit the glyph completely.
  - Otherwise, compute the transformed destination rectangle and blit with the scale.
- Compute the glyph destination position by using unscaled glyphs to preserve the metrics, but use the scaled font.

Caching Sub-Scenes

- The CachingSubSceneNode is container that draws its child on a texture, which is then reused on subsequent redraws.
- Enabling the generation of this caching node on a Morph requires overriding the #shouldCacheSceneGraphSurface to return true.
- Pixel perfect should be attempted to avoid text blurring.
- Caching enabled for Windows, Menus, Rubric and tables.
- This caching textures typically end with premultiplied alpha, so an adequate blending mode is required here:
  - Premultiplied Alpha Over: glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA)
  - Composite Alpha Over: glBlendFuncSeparate(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA, GL_ONE, GL_ONE_MINUS_SRC_ALPHA)
- Subpixel antialiased text does not behave well on transparent backgrounds here.

Caching Sub-Scene



```
1 World scaleFactor: 5.0
```

Questions?