

Introduction

The aim of this project is to develop a collection of applications which can render custom graphics. Inspiration taken from the VizRT suite of applications. Currently collection consists of Chroma Viz, Hub, Engine and Artist. Chroma Viz, Hub and Artist are built in Golang and are contained in the [Chroma Viz](https://github.com/jchilds0/chroma-viz) repository on Github. [Chroma Engine](https://github.com/jchilds0/chroma-engine) repository.

```
<source src="https://github.com/jchilds0/chroma-viz/raw/main/data/demo.m
```

```
</video>
```

Chroma Viz

Chroma Viz manages templates at a high level, and issues commands to Chroma templates from Chroma

of all templates in the hub and sends this list to Chroma Viz.

clicking on the template in the Templates tab. Chroma then send a request to Chroma Hub for this template. Chroma implements a REST API for accessing assets such as templates. Pages form shows, which can be saved to/loaded from disk for future

alt="Templates">

Engine instances, using tcp sockets to communicate. Each connection is either a Engine or Preview connection. When user opens a page to edit by double clicking on the page, or by saving changes made to the page with the save button, the page is sent to all connections with the preview type. Chroma provides a C library which can create a GtkGLRender widget, and using cgo, Chroma Viz creates a Chroma Engine preview window. Chroma Viz sends pages to layer 0 of the preview window, so pages switch as expected which changing between pages. The actions at the top of the editor panel, \$ exttt{Take On, Continue} \$


Chroma Engine instances with the Engine type.

Take On animates from Keyframe 1 to Keyframe 2.

Continue runs from the current Keyframe to the next Keyframe.

Take Off runs from the second last Keyframe to the last Keyframe.

Chroma Viz encodes the attributes of a page and sends it to

Chroma templates attributes before rendering the graphic. 

Chroma Engine

Chroma Engine renders graphics requests from Chroma Viz. At core, Chroma Engine creates a GtkGLRender widget which renders graphics.

Viz to create a preview window, and a binary, which creates a standalone GTK application which only contains the GtkGLRender.

and requests all templates in the Hub. This Engine can build its own database containing each template that could be received from Chroma Viz. This needing to allocate resources for each template, but the benefit is we don't need to allocate memory at run time when we receive a graphics options would be allowing the user to load a subset of templates currently in use, and requesting any new templates on the fly from Chroma Hub as needed. Chroma graphics using

receives a string of data from a Chroma Viz instance. A name-attribute format purposes.

layer, template id, and action. Then by an integer, followed by a list of attributes for the geometry.

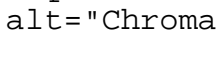
geometry of the target template. Chroma geometry

set, for each pixel of the geometry we check if every parent geometry also has a pixel at this point. To utilise OpenGL stencil buffers, keeping a buffer for each parent and then drawing the geometry where we can pass through all

buffers, restricting geometry tree depth to a maximum of 8.

Image assets are also contained in Chroma Hub, so before we can render an image we request it from Chroma Hub. Chroma send 4 bytes with the length of the image, followed by the image as a raw as well as decoding the png to extract the pixel data using libpng.

currently stored image id, we reuse this file instead of requesting the image again from Chroma Hub. After graphics request and any image assets, Chroma Engine computes the keyframes for the template. This position calculations.

Chroma Artist curve to control the animation timing. Finally the page is rendered to the screen using OpenGL. 

Chroma Hub

The purpose of Chroma Hub is to synchronize the graphic templates used by Chroma Viz and Chroma Engine instances. Chroma Hub also stores any assets needed by the templates such as images, currently only in png format. Chroma database, which currently needs to be setup with the schema in `hub/chroma_hub.sql` package for simplicity to write and read a json format of the database to a assets updating/retrieving application which makes POST requests, to import/update templates and requests to retrieve assets.

Chroma Artist

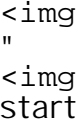
Chroma Artist provides a UI for designing templates which can be imported to Chroma Hub and used by Chroma Viz. The difference between Chroma Viz and Chroma Artist is Chroma Artist is used to manipulate the geometry hierarchy of a template, and create Keyframes for the template. In discussion of Chroma Engine, we omitted the discussion of relative coordinates. graphics, each geometry has a parent geometry. This geometries a tree structure, and position of a geometry is relative to the position of the parent geometry. Chroma gives an easy interface to specify this tree structure, which Chroma Engine rebuilds to calculate the absolute positions. An example of this functionality is a simple lower frame super, which contains a rectangle for the background, two text geometries parented to the rectangle, and a circle as a logo placeholder also parented to the rectangle. The set the default position, width and height of the background rectangle, aswell as the position of the text. Since geometries and circle are parented to the rectangle, to move the graphic we only need to change the rectangle position. In Chroma Artist, the following image shows an example of this graphic.

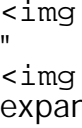
```
![[keyframe](/assets/chroma-graphics/keyframeartist.png)]
```

Currently the rectangle is static and the width of the rectangle needs to be updated when the text changes. Keyframing allows us to animate the graphic and have the width of the rectangle be linked to the text width. The based on a directed acyclic graph. We case of no keyframes which is used to calculate the absolute positions from relative position. This construct keyframes. represents some attributes of the graphic we described above,

```
 
```

The position of geometry 4 depends on the x position of geometry 1 (it's parent) and its own relative x position. We dependency with an arrow from the x position of geometry 4 to the x position of geometry 1 and the relative x position of

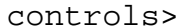
geometry graph, which is a directed acyclic graph.  At node of this graph, we want to compute the value of the node by combining the values of the dependencies in some way. To this we store a function at each node, in the case of relative positions this function evaluates the sum of the value for each child node of the current node. But node only once each child node has been evaluated. This is known as topological sorting. This William Fiset describes topological sorting of directed acyclic graphs ([Topological Sort](https://youtu.be/eL-KzMXSXXI?si=mG9)) Tj ET BT

adding the following dependencies, and adding a function to the rectangle node which evaluates the maximum of the value of each child node.  The keyframing process in Chroma Artist consists of two objects, Frames and fixed number n , and contain a collection of keyframes. For each frame, we create the table of attributes for each geometry show evaluation functions to the graph. When attributes are interpolated linearly between the frames. Current types of keyframes are

Set Frame: Set the value of an attribute of a geometry in a specific keyframe. In the graph this updates the value of a node and doesn't add any dependencies.

User Frame: Similar to set frame, except the value from the template or page when the graphic is animated on is used. This adds an edge which points to the attribute values set by the user, and is the default for non keyframed attributes.

Bind Frame: Use a value computed in a keyframe. This adds an edge to another node in the graph, and the current node simply takes the value of the single child node.

Additionally we can set a keyframe to be an expand keyframe, currently only supported for User Frame rectangles and attributes width or an edge to the upper x position and upper y position for width and height respectively for each child geometry, and evaluates the maximum of all child nodes. The keyframes is they cannot create cycles in the graph. Doing makes evaluating them ambiguous, so Chroma Engine terminates if it receives a template with a cycle in the keyframe graph. Putting these together we can create the following graphic. 

```
<source src="https://github.com/jchilds0/chroma-viz/raw/main/data/artist
```

```
</video>
```

```
href="/assets/chroma-graphics/style.css">
```