

# Introduction

The aim of this project is to develop a collection of

Currently this collection consists of Chroma Viz, Hub, Engine

Chroma Viz, Hub and Artist are built in Golang and are contained in the [Chroma Viz](https://github.com/jchilds0/chroma-viz) repository on

Chroma Engine is built in C and is contained in the [Chroma

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

## Chroma Viz

Chroma Viz manages templates at a high level, and issues

Chroma Hub collects the template IDs of all templates in the

```
<img src="/assets/chroma-graphics/templates.png"
```

Pages can be created from templates, by double clicking on the

Chroma Viz will then send a request to Chroma Hub for this

Chroma Hub implements a REST API for accessing assets such as

Pages form shows, which can be saved to/loaded from disk for

Chroma Viz can connect to any number of Chroma Engine

When the 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

Chroma Engine provides a C library which can create a GtkGLRender widget, and using cgo, Chroma Viz creates a Chroma

Chroma Viz sends pages to layer 0 of the preview window, so

The actions at the top of the editor panel, \$ exttt{Take On,

\$ exttt{Take Off} \$, send pages to connected Chroma Engine

Take Off runs from the second last Keyframe to the last

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

Chroma Engine parses the message and updates the templates

```
<img src="/assets/chroma-graphics/chroma-viz.png" alt="Chroma
```

## Chroma Engine

At its core, Chroma Engine creates a GtkGLRender widget which

We compile both a library, which is used by Chroma Viz to create a preview window, and a binary, which creates a

On startup, Chroma Engine connects to Chroma Hub and requests

This is done so Chroma Engine can build its own database containing each template that could be received from Chroma

This has the added cost of needing to allocate resources for each template, but the benefit is we don't need to allocate

A middle ground between these two 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

To render a graphic, first Chroma Engine receives a string of

The string contains a header with the format version, layer,

Then each geometry, specified by an integer, followed by a list

As we parse the string, we set the values for each geometry of

Each geometry has a masking attribute, and if set, for each pixel of the geometry we check if every parent geometry also

To achieve this we utilise OpenGL stencil buffers, keeping a buffer for each parent and then drawing the geometry where we

GTK restrictions mean we can only have 8 stencil buffers,

Image assets are also contained in Chroma Hub, so before we can Chroma Hub first send 4 bytes with the length of the image, Then we store this data for later use, as well as decoding the In later render calls, if the image id matches the currently stored image id, we reuse this file instead of requesting the After receiving the graphics request and any image assets,

For smooth animations, we use a bezier curve to control the Finally each geometry in the page is rendered to the screen  
<img src="/assets/chroma-graphics/chroma-engine.png"

## **Chroma Hub**

The purpose of Chroma Hub is to synchronize the graphic Chroma Hub also stores any assets needed by the templates such Chroma Hub wraps a SQL database, which currently needs to be

Golang package for simplicity to write and read a json format Chroma Hub implements a REST API for updating/retrieving Chroma Artist is currently the only application which makes Chroma Artist/Viz and Engine using GET requests to retrieve

## **Chroma Artist**

Chroma Artist provides a UI for designing templates which can

The key difference between Chroma Viz and Chroma Artist is Chroma Artist is used to manipulate the geometry hierarchy of a

In the discussion of Chroma Engine, we omitted the discussion To enable easier manipulation of graphics, each geometry has a This gives the geometries a tree structure, and position of a

Chroma Artist gives an easy interface to specify this tree structure, which Chroma Engine rebuilds to calculate the

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

The designer could set the default position, width and height of the background rectangle, aswell as the position of the

Since the text geometries and circle are parented to the rectangle, to move the graphic we only need to change the

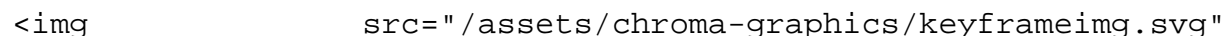
In Chroma Artist, the following image shows an example of this

Currently the rectangle is static and the width of the

Keyframing allows us to animate the graphic and have the width

We begin with the simplest case of no keyframes which is used

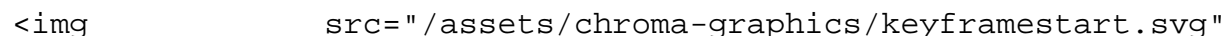
Consider the following table which represents some attributes

 src="/assets/chroma-graphics/keyframeimg.svg"

The x position of geometry 4 depends on the x position of

We indicate this dependency with an arrow from the x position of geometry 4 to the x position of geometry 1 and the relative

Continuing for all geometries we form the following graph,

 src="/assets/chroma-graphics/keyframestart.svg"

At each node of this graph, we want to compute the value of the

To do this we store a function at each node, in the case of relative positions this function evaluates the sum of the value

But we need to evaluate each node only once each child node has

This excellent video by William Fiset describes topological

We can make the rectangle width dynamic by adding the following dependencies, and adding a function to the rectangle node which

```
<img src="/assets/chroma-graphics/keyframeexpand.svg"
```

The keyframing process in Chroma Artist consists of two Frames have an index 1, 2, ... up to some fixed number  $n$ , For each frame, we create the table of attributes for each A Keyframe is a way to add dependencies and evaluation When animated, geometry attributes are interpolated linearly

Set Frame: Set the value of an attribute of a geometry in a specific keyframe. In the graph this updates the value of a

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

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

Additionally we can set a keyframe to be an expand keyframe, currently only supported for User Frame rectangles and

This is what we used above, we add 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

The only restrictions on keyframes is they cannot create cycles

Doing so makes evaluating them ambiguous, so Chroma Engine terminates if it receives a template with a cycle in the

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

```
<link rel="stylesheet"
```

