Animation Generation Language

by

Chaitanya Sunkara

A Creative Component submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Dr. Simanta Mitra, Co-major Professor
Dr. Gurpur Prabhu, Co-major Professor

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this creative component. The Graduate College will ensure this dissertation/thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2023

DEDICATION

I would like to dedicate this creative component to my mother Rama Devi, my father Ramesh, and my sister Anusha without whose support I would not have been able to complete this work.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

NOMENCLATURE

Libraries

GSAP GreenSock library

SVG.js SVG.js library

Terminology

AGL Animation Generation Language

GPAL General Purpose Animation Language

high-level user One who uses commands

IDE Integrated Development Environment

low-level user One who defines and implements commands

SVG Scalar Vector Graphics

ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this creative component. First and foremost, Dr. Simanta Mitra for his proposal, guidance, patience and support throughout this research and documentation. His insights and words of encouragement have often inspired me and renewed my hopes for completing my graduate education. I would also like to thank my committee members for their efforts and contributions to this work: Dr. Gurpur Prabhu.

ABSTRACT

Research has shown that people learn quickly and understand things better when presented visually rather than verbally. There are many JavaScript based animation tools and libraries available in the market. However, most of them are restricted to a particular domain, have predefined programs to animate and cannot be used to build on top of existing functionality. Animating SVG elements is one such great way to represent things visually. This project proposes a new language for creating dynamic, and interactive animations with SVG.

The language is designed to simplify the process of animating SVG elements, with a focus on ease-of-use, flexibility, and re-usability. Key features of the language include support for a wide range of animation types, including scaling, and rotating, as well as advanced capabilities such as custom easing functions and timeline management. The language also supports some special features like views and commands. Additionally, a brand new UI was developed to enable the creation of high-quality animations with less manual effort, while also allowing for more creative control over the animation process.

We demonstrate the effectiveness of our language by generating a range of animations, including generic animations, git animations and special effects. Our results show that our language outperforms existing approaches in terms of both quality and efficiency, while also enabling more creative exploration of the animation space. Overall, this new language offers a powerful and intuitive solution for creating engaging and visually stunning animations with SVG, making it a valuable tool for instructors.

CHAPTER 1. Introduction

JavaScript has become an essential part of modern web development, and it continues to evolve with new libraries and frameworks being developed every day. Scalable Vector Graphics (SVG) is a popular format for creating high-quality graphics and animations on the web due to its flexibility and versatility. SVG graphics are resolution-independent and can be scaled without losing quality, making them an ideal choice for creating animations. However, creating complex SVG animations using traditional JavaScript methods can be time-consuming and challenging.

To address this issue, a range of JavaScript libraries have been developed, each offering different features and benefits. However, all these libraries target JavaScript developers but not end users. In other words, a naive user cannot animate SVG elements using these libraries because he/she doesn't have the extensive knowledge of JavaScript.

In this paper, we introduce a new language, Animation Generation Language (AGL) for SVG animation, which we believe offers a unique and intuitive approach to creating dynamic SVG animations. The new language is designed to provide high-level (naive) and low-level (developer) users with a straightforward and user-friendly way to create complex SVG animations, without requiring extensive coding knowledge. The language's simple syntax and intuitive API make it easy to customize animations and create engaging effects. Additionally, the language is optimized for performance, ensuring fast load times and smooth animations using cutting-edge React framework, SVG.js and GreenSock libraries.

One of the key benefits of the new language is its versatility. It offers a range of animation options and color transitions, which can be combined to create intricate and visually appealing effects. The language also supports a range of SVG attributes, giving designers and developers greater control over the animation process.

In the following sections, we will provide a detailed overview of the language's features, discuss its benefits, compare it with existing approaches and provide examples of how it can be used to create engaging and interactive SVG animations for the web.

CHAPTER 2. Background and Related Work

2.1 Background

Most of the animation tools and libraries can be categorized into two groups - simple and complex. Simple animation tools doesn't require an extensive coding knowledge. These tools are generally easy to use, interactive, and user friendly. However, they have limited capabilities. A good example of this type is PowerPoint. It can be used to create primitive animations by specifying sequence of objects, duration, delay, what triggers the animation. The drawbacks are you don't have control over the properties of object like scale, color and mainly you don't have programmatic access of animation.

Complex animation libraries like D3.js, are much more powerful and can be used to create complicated animations. They overcome the shortcomings of simple animation tools. They have control over all the attributes of object and can programmatically perform the animations.

However, they have their own disadvantages. Most of them are developer focused, non interactive, not user friendly and require extensive coding knowledge. For any action you have to create a library object and add relevant attributes. There are no commands for any action so it is time consuming and tedious to create animations.

2.2 Related Work

General Purpose Animation Language (GPAL) that was developed by (1) tried to fulfill the shortcomings of both simple and complex animation tools. GPAL is a script based animation language with limited capabilities. It has 2 scripts, one is used to draw objects and the other is to perform moderate level animations. It supports few data types, expressions, operators, global

variables and basic functions. On the animation side, it supports only move operation and styling is limited to stroke and border. It supports high-level commands but has to be specified in json file.

However, GPAL is loosely defined and restricted in terms of language and animation capabilities. For example, it does not support all data types, statements, conditions, loops, oop, functions with parameters and return statement, etc... On the animation side, it doesn't support all attributes like scale, rotation, skew, duration, delay, etc... In terms of svg, it doesn't support path, polyline and can't update attributes of elements. Moreover, the UI is not user friendly - there's no visible pointer information, can't use drawing script in animation script. Mainly, it is not flexible and versatile to extend the capabilities of language. For instance, it's not possible to add a new plugin that implements click functionality to the language without any modification to the language.

So, there is need for an animation language that is interactive, user friendly, flexible, supports all features of the language, supports all animation properties, and powerful enough to support both naive users and users with programming experience. Animation Generic Language (AGL) was developed to fill this void. It based on JavaScript and has a single script for drawing and animating elements, supports all language features and animation properties including scaling, rotating along with advanced features like easing functions. It has some special features like views and commands. Furthermore, a brand new UI is developed to facilitate the creation of animations with less effort and options to visualize them.

2.3 Why AGL and not GPAL v2

Although, AGL and GPAL are trying to solve the same problem, fundamentally there's lot of differences between them on how they are designed and implemented. I believe, there are many poor choices made in GPAL. Firstly, GPAL follows the philosophy of developing everything in house. It implements its own parser in JavaScript to read the drawing and animation script. This

is not a wise decision because it is impossible to cover all user inputs. Secondly, it doesn't use any frameworks or libraries. Leveraging existing open source solutions can greatly reduce the effort and helps to focus only on the core functionality. Thirdly, it uses html in JavaScript to create and update the svg element's attributes. This will result in poor performance, vulnerabilities and hard to maintain code base, could have been easily avoided by simply using JavaScript api's. Lastly, the code base is not modular and doesn't reuse components. This results in redundancy and duplicates. Because of these reasons, it is preferred to start from scratch rather than pursuing GPAL v2.

2.4 References

[1] Calpakkam, V. K. (2019). General purpose animation language. Master's thesis, Iowa State University.

CHAPTER 3. Design and Implementation

3.1 Overview

The basic idea is that there will be two types of users low-level and high-level user. Low-level user will write the animation script by defining certain commands and its implementation using AGL. High-level user will use these commands to visualize the animations. For example, lets consider git scenario, a low-level user will create the command say "git commit" and implement its functionality of creating new commit object, moving the head to new commit id. When a high-level user uses the command "git commit" it will invoke the functionality implemented by low-level user and performs the animation. If there are no commands in animation script, then AGL should render the animation without any input from user.

Figure 3.1 shows the IDE used for AGL.

3.2 Approaches Tried

There are in total 5 approaches that I've experimented with before choosing the right one. The first approach I've tried is using only JavaScript like GPAL but using its APIs to perform the animation rather than updating html. Although, there's nothing wrong with this approach, pretty soon I've realized this is already implemented by many libraries and no point in reinventing the wheel. The second approach I've experimented is using three.js library. It was good for supporting primitive animation attributes but lacks advanced features like timeline functionalities.

Next I've tried combining React framework and snap.js. React has greatly reduced the effort needed on UI front, because it has great libraries that support various UI features. Snap.js has support for all animation features however the performance was poor and api documentation is not great. Later I've experimented with React and GreenSock (gsap). GreenSock is cutting-edge

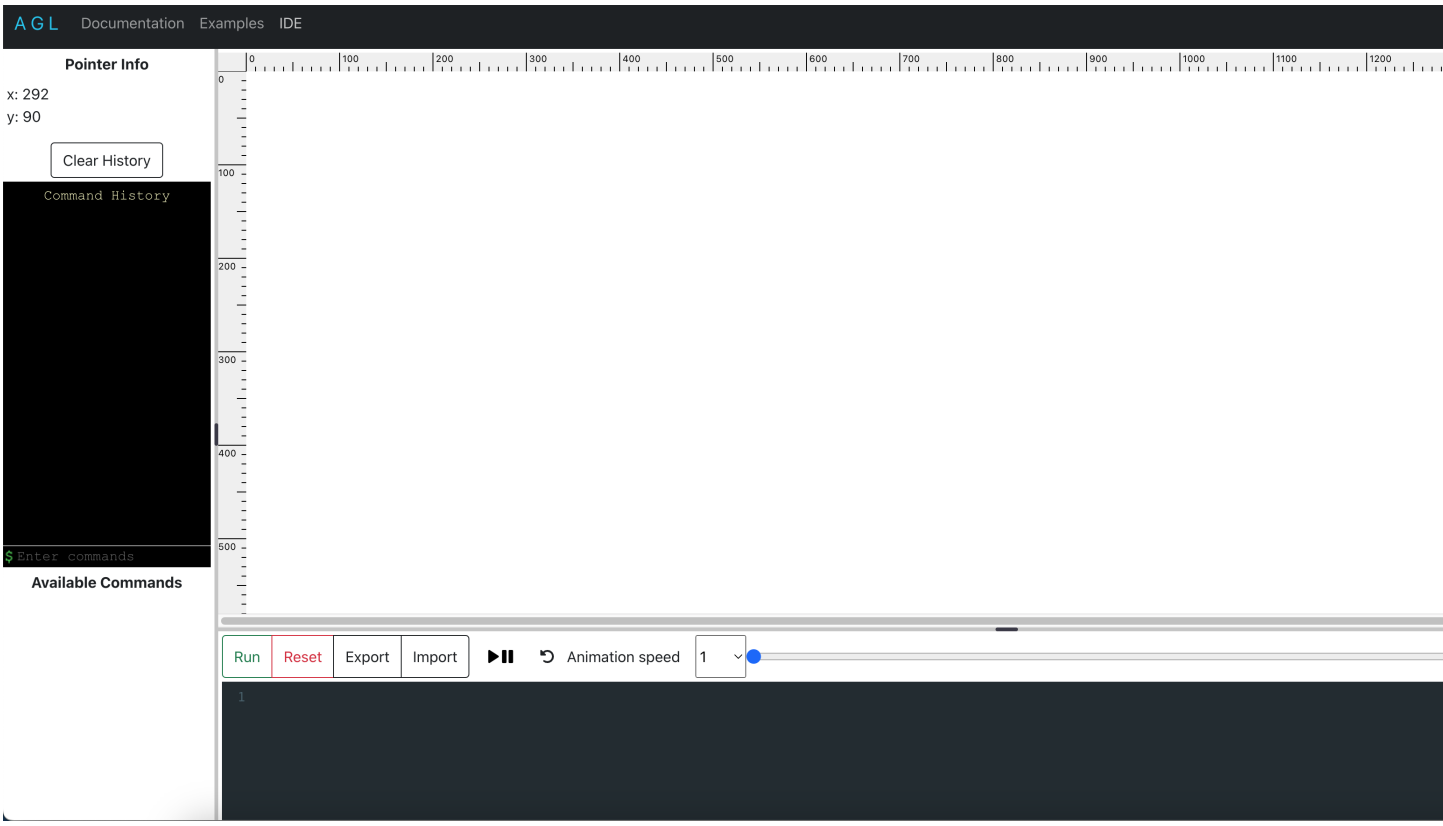

Figure 3.1 IDE

and very good at supporting advanced features. However, it lacks support for basic svg functionalities like creating a rectangle.

Finally, I've landed on combining React with GreenSock and SVG.js. This combination gave desirable results because we are leveraging only the best features from each of these libraries. So, SVG.js is only responsible for creating the svg elements and updating its attributes. GreenSock is solely responsible for animating the svg elements and timeline functionalities. React has made it possible for the UI to be much more modular, component driven and optimized.

3.3 Architecture

The high level architecture of AGL is outlined in Figure 3.2. It comprises of 4 components.

3.3.1 High-level commands/input

User will input high-level command that invokes a function defined in low-level AGL script. It is also possible to pass dynamic parameters from high-level commands to that function as an argument. User input can be in other forms as well like clicking on a object.

3.3.2 Low-level AGL script

Low-level script contains the animation definition. It describes what attributes to update, what animation sequence to follow etc... using AGL script.

3.3.3 Convert to JavaScript

At run time AGL script is converted to JavaScript using babel library and executed.

3.3.4 Render UI

JavaScript with the help of libraries GSAP and SVG performs the animation and renders it on the UI.

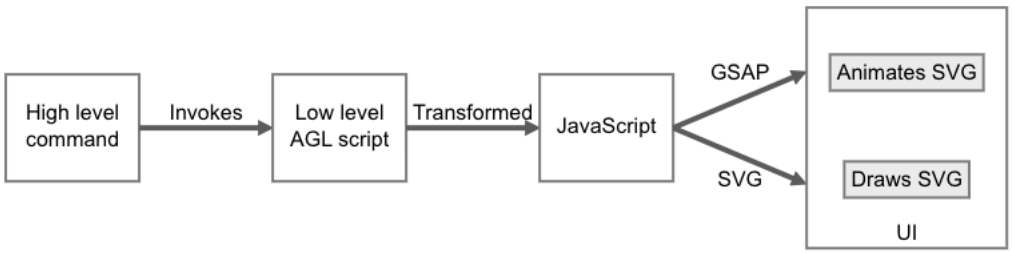

Figure 3.2 AGL Architecture

Table 3.1 AGL APIs

Category	Element	Syntax
	Rectangle	RE(id, width, height, x, y, rx, ry, stroke, strokeWidth, fill)
	Circle	CI(id, cx, cy, r, stroke, strokeWidth, fill)
	Ellipse	EL(id, cx, cy, rx, ry, stroke, strokeWidth, fill)
	Line	LI(id, x1, y1, x2, y2, stroke, strokeWidth)
Draw	Text	TE(id, x, y, text, fontSize, fontFamily, fill)
	Polygon	POG(id, points)
	Polyline	POL(id, points)
	Path	PA(id, d, stroke, strokeWidth, fill)
	Group	GR(id, ...groupIds)
	RectangleText	RETE(id, x, y, text, padding, fontSize, fontFamily, fill, textAnchor, allignmentBaseline)
	CircleText	CITE(id, text, x, y, padding, fontSize, fontFamily, fill, textAnchor, allignmentBaseline)
~~SpecialDraw~~	Arrow	AR(id, x1, y1, x2, y2, stroke, strokeWidth)
	PolyArrow	PAR(id, points)
	DoubleArrow	DAR(id, x1, y1, x2, y2, stroke, strokeWidth)
	PolyDoubleArrow	PDAR(id, points)
	ChangeAttributes	ATTR(id, attribute1:value1, ...(attribute2:value2))
	AddToGroup	ADDGR(id, ...groupIds)
	UnGroup	UNGR(oldGroupId, newGroupId)
~~Manipulate~~	InsertBefore	INSBF(id1, id2)
	InsertAfter	INSAF(id1, id2)
	Hide	HIDE(id)
	Show	SHOW(id)
	Remove	REM(id)
	FROM	FROM(id, message, attribute1:value1, ...(attribute2:value2))
	TO	TO(id, sync, message, attribute1:value1, ...(attribute2:value2))
Animate	FROMSYNC	FROMSYNC(id, sync, message, attribute1:value1, ...(attribute2:value2))
	TOSYNC	TOSYNC(id, sync, message, attribute1:value1, ...(attribute2:value2))
	UpdateAttributes	SET(id, attribute1:value1, ...(attribute2:value2))
	Command	$cmd = { functionName(); }
	Command with parameters	$cmd #param1 #param2 = { functionName(#param1, #param2); }
Misc	Views	VIEW(”id”);
	Variables	@viewId.variableName
	Click	CLICK(id, functionName)

3.4 AGL APIs

3.4.1 Syntax

AGL syntax is simple, intuitive and supports multiple variations. However, it is case-sensitive. All commands are upper-cased in order to distinguish them easily and usually start with the first two letters of element. All strings should be surrounded by quotes like "Sample". If you need to use quotes in string, it should be escaped like "Sa\"m"\ple". It is recommended to provide an id for the elements you create so that it can be uniquely identified. All ids should start with an alphabet and can contain alpha numeric underscores. If you don't want to provide an id, pass an empty string like "" and AGL will create an id. However this id is only visible by inspecting the element. AGL supports styling in all formats, example colors can be provided as "red", "#ff0000", or "(255,0,0)". Table 3.1 shows all the low-level commands that are grouped into 5 categories.

1. Draw

    - These commands are used for creating svg elements.
    - Example: RE("rect1", 200, 100, 50, 50, 0, 0, "red", 3, "black")
    - None of the arguments are mandatory. You can just use RE() and it creates a rectangle with default values.
    - Supports overloading of arguments from left to right. Uses default values for missing arguments

      RE("rect1")

      RE("rect1", 300)

      RE("rect1", 300, 200)
    - If you don't want to provide a value for an argument and prefer default values, pass empty string "". For example you want to change the position of rectangle but don't want to give width and height.

      RE("", "", "", 250, 100)

- Grouping of elements takes any no.of ids as arguments
  GR(”group1”, ”rect1”, ”circle1”, ”rect2”)

2. SpecialDraw

   - These are some customized commands that help in creating most frequent objects.
   - Example: Creating a text and placing it in a rectangle.
   - Instead of using two separate commands. RETE will directly create a text and dynamically create a rectangle around the text based on text length.
   - Spcae between text and rectangle can be customized using padding parameter.

3. Manipulate

   - These commands are used to change/update the attributes of an element.
   - Example: ATTR(”rect1”, ”stroke:blue”, ”width:300”)
   - Hide doesn’t delete the object, just makes it invisible.
   - Remove completely deletes the object and can’t be recovered.

4. Animate

   - These commands are used to animate the objects.
   - FROM command starts the animation from specified state and reaches the creation state.
   - TO command starts the animation from creation state and reaches the specified state.
   - FROM and TO commands perform animations sequentially.
   - FROMSYNC and TOSYNC commands are similar to FROM and TO but perform animations in parallel.
   - sync parameter specifies the position parameter like ”<” - start of previous animation, ”>” - end of previous animation, ”=” - same as timeline.

- message parameter can be used to display information about the animation. Message will be shown in command pane below pointer information.
- Example: FROMSYNC(”rect1”, ”=”, ”sample animation”, ”scale:0”, ”transformOrigin:right”);

5. Misc

- These commands handle some of the exclusive features of AGL.
- $cmd is used to define high-level commands and is mapped to a function in the script. When the high-level command is used, corresponding mapped function is invoked.
- $agl init is an initialize command, if present it will be executed for every run button click.
- #param is used to read parameters used with high-level command and can be forwarded to mapped function. Order of parameters can't be changed.
  Example: $git branch # = sampleFn(#)
- Views are like private access modifiers. They can be used to restrict the scope of objects to a particular view.
- By default all variables have global scope. But you can restrict its scope by linking it to a view.
- CLICK command can be used to invoke a function when a particular object is clicked.

Rest of the features of the language like datatypes, variables, expressions, operators, conditions, statements, functions, loops etc... will be same as in JavaScript.

Since all the animation script is eventually converted to JavaScript, all native JavaScript capabilities and functionalities can be leveraged in animation script. For instance, if you want to debug the script, although there's no api defined in AGL, you can still use *console.log*() or *alert*() in the animation script. This flexibility makes AGL much more powerful and removes any

shortcomings of the language. Additionally, this opens the gate to developer community, to extend the capabilities of AGL by creating plugins and extensions without any change to AGL.

CHAPTER 4. Results

4.1 Comparison of GPAL and AGL

In this chapter we compare AGL with GPAL in terms of svg features, animation capabilites, language features, functionality, UI ease of use. It is evident from Table 4.1 that AGL overcomes the limitations of GPAL.

Table 4.1: GPAL vs AG

Begin of Table		
	GPAL	**AGL**
Libraries	jQuery	React SVG.js GSAP
Philosophy	Develop everything in-house using plain html and javascript	Don't reinvent the wheel, integrate the best frameworks that are suited for the job i. React for UI ii. SVG.js for drawing iii. GSAP for animating
Scripts	Two separate scripts i. Drawing ii. Animation Must have "function_main" in Animation script	Single script for drawing, animating, mapping commands

Continuation of Table 4.1		
	GPAL	**AGL**

Drawing	i. Doesn't support polyline and path
	ii. Can't change attributes/properties
	iii. Can't hide/remove elements
	iv. Doesn't support Special Drawing elements
	v. Doesn't support overloading of commands
	circle(100,100,30);
	circle(100,100, 30, c1);
		i. Supports all svg elements
		ii. Can change attributes/properties
		iii. Can hide/remove elements
		iv. Supports some special elements for convenience - RectangleText, CircleText, Arrow, PolyArrow, DoubleArrow, PolyDoubleArrow
		v. Supports overloading of commands
		CI(); CI("c1"); CI("", 100); CI("c1", 100, 100, 30, "#888888", 5 ,"#EEEEEE");

Animation	Doesn't actually perform animation just updates attribute value. So element disappears at 'A' and appears at 'B' instead of moving from 'A' to 'B' i. Styling - Limited to stroke/border (red, green, black), stroke width on(s1), select(s1), off(s1), pulse(s1) ii. Attributes - Limited to x move(s1,100); iii. Animation - doesn't support animation properties	Performs the animation - element actually moves from 'A' to 'B' smoothly. i. Styling - Supports all types of styling - stroke, stroke width, fill, opacity... ii. Attributes - Supports all attributes - x, y, scale, rotation, skewX, skewY, transformOrigin... iii. Animation - Supports all animation properties - duration, delay, relative timing

Continuation of Table 4.1

	GPAL	**AGL**

Language	Loosely defined and limited capabilities
	i. Data types - doesn't support boolean, null, objects,integer, float, character, strings
	ii. Variables - only global scope
	var(x=1000);
	iii. Expressions - basic mathematical operations performed using eval function in js
	exp(x=x+10);
	iv. Assignment - basic
	var(x=10,y=0); exp(x=x+20); var(y=x);
	v. Functions - doesn't support parameters, return
	function_one(){}
	vi. Statements, Conditions, Loops, OOP - doesn't support
		All javascript language features are supported
		i. Data types - Boolean, Null, Undefined, Number, BigInt, String, Symbol, Objects
		ii. Variables - Local, Global
		iii. Expressions - all expressions
		iv. Assignment - all assignments
		v. Functions - full support - supports parameters, return
		vi. Statements, Conditions, Loops, OOP - full support

IDE	Not user friendly and too many alerts for displaying information i. NO ruler/scale for drawing ii. NO pointer information iii. Complex import functionality iv. Basic timeline	NO alerts and much more user friendly i. Ruler for easy visualization ii. Dynamic pointer information iii. Simple import iv. Advanced timeline with option to go forward and backward

Continuation of Table 4.1		
	GPAL	**AGL**
Functionality	i. NO concept of commands ii. NO concept of views iii. Doesn't support click functionality on elements iv. Doesn't support javascript in any of the drawing or animation scripts	i. High-level commands and mapping of high-level commands to functions ii. Supports views and variables tied to views. iii. Supports click functionality iv. Supports full javascript console.log(); alert(); document.getElementById()

Continuation of Table 4.1

	GPAL	**AGL**
Working	i. User does the low-level scripting for both drawing and animation ii. Parses the drawing script and renders them by modifying html iii. Parses the animation script, mainitains a queue of animations and changes the attributes of elements	Script is made up of javascript, low-level commands and high-level commands Low-level commands are pre-defined javascript functions and of two types a. Drawing - CI, RE, LI b. Animation - FROM, TO, FROMSYNC, TOSYNC If low-level command is of type drawing then it forwards to SVG library which uses javascript to render the elements If low-level command is of type animation the it forwards to GSAP library which uses javascript to animate the elements High-level commands are mapped to functions in the script i. User does the low-level scripting ii. Stores a list of high-level commands and its mapping iii. Removes high-level commands from the script iv. Combines script with low-level command definitions v. Convert the combined script to javascript and execute it

End of Table

4.2 Evaluation with other tools

AGL is greatly inspired by the existing animation tools and libraries in the market. Thus, it makes sense to compare AGL with existing solutions. In the below table we've compared AGL with Visual Algo, Git with D3, Loupe and Power point in terms of capabilities and functionalities. It can be seen from Table 4.2 that AGL is capable of animating all use cases.

Table 4.2 Other tools vs AGL

	Visual Algo	**Git with D3**	**Loupe**	**Power Point**	**AGL**
Domain	Covers Data structures and algorithms	Covers git commands	JS call stack	ALL	ALL
Dynamic	YES. Can only add elements	YES. Can only add elements	Yes. Can only add elements	NO	YES. Depends on implementation of commands
Functionality Extendability	NO	NO	NO	NO	Can add any extension that is in javascript
Timeline	YES	NO	NO	NO	YES

CHAPTER 5. Conclusion, References and Appendix

5.1 Conclusion

In this project, a new language is developed for creating and animating SVG elements. The language provides a straightforward and user-friendly way to visualize SVG animations. It has simple syntax and intuitive api to create complex SVG animations without requiring extensive coding knowledge. It covers all scenarios and is not restricted to a domain. It is optimized for performance, fast load times and seamless animations across all devices.

AGL supports all svg elements, animation properties and features of a language. It also supports some special features like high-level commands and views. It is much more powerful, versatile and flexible than its predecessors. Moreover, it facilitates the use of JavaScript in low-level scripting, that allows the developer to extend the capabilities of language by creating plugins and extensions.

In future, the language can be extended to support import statements and mouse driven api's like drag and drop, hover, etc... Supporting relative values for SVG creation and animation. Going beyond SVG, language can be extended to support images that can be uploaded. Integrating with other libraries like D3.js that are focused on specific domains such as data visualization, charts, graphs, maps etc... Theoretically, it can implement all the api's of JavaScript.

In conclusion, Animation Generation Language is an exciting addition to the world of JavaScript animation. With AGL, developers can create animations that are both beautiful and performant, ensuring a great user experience on any device.

5.2 References

[1] Calpakkam, V. K. General purpose animation language. Master's thesis, Iowa State University, 2019.

[2] Visualizing git concepts with d3. [online] https://onlywei.github.io/explain-git-with-d3/.

[3] Gsap. [online] https://greensock.com/gsap/.

[4] Visualizing javscript callbacks. [online] loupe.

[5] Build your own interactive javascript playground. [online] https://krasimirtsonev.com/blog/article/build-your-own-interactive-javascript-playground.

[6] Svg.js. [online] https://svgjs.dev/docs/3.0/.

[7] Visualizing data structures and algorithms. [online] https://visualgo.net/en.

5.3 Appendix A: Examples

5.3.1 Example 1: git commit and branch

```
function load(){
    VIEW("v1");
    TE("",80,50,"Local␣Repository");
    TE("",100,80,"Current␣Branch:␣master");
    @v1.aid = 0;
    @v1.arx1 = 25;
    @v1.arx2 = @v1.arx1-45;
    @v1.ary = 200;
    @v1.cid = 0;
    @v1.cx = 50;
    @v1.cy = 200;
    @v1.tid = 0;
```

```
  @v1.tx = @v1.cx;
  @v1.ty = @v1.cy+40;
  @v1.trsx = 0;
  @v1.currentBranch = "master";
  @v1.currentCommitId = "e137e9b";
  @v1.branches = ["master"];
  @v1.commitIdBranches = {};
  GR("arrows");
  GR("circles");
  GR("commits");
  AR("a"+@v1.aid,@v1.arx1,@v1.ary,@v1.arx2,@v1.ary);
  ADDGR("arrows", "a"+@v1.aid);
  CI("c"+@v1.cid,@v1.cx,@v1.cy,"","#339900","","#CCFFCC");
  ADDGR("circles", "c"+@v1.cid);
  TE("t"+@v1.tid,@v1.tx,@v1.ty,@v1.currentCommitId);
  ADDGR("commits", "t"+@v1.tid);
  FROMSYNC("a"+@v1.aid, "=", "", "scale:0", "transformOrigin:right");
  FROMSYNC("c"+@v1.cid, "=", "", "scale:0", "transformOrigin:center");
  RETE("master",@v1.tx,@v1.ty+40,@v1.currentBranch);
  RETE("head",@v1.tx,@v1.ty+80,"HEAD");
  ATTR("master-rect", "fill:#FFCC66", "stroke:#CC9900");
  ATTR("head-rect","fill:#CCFFCC", "stroke:#339900");
  CLICK("c"+@v1.cid, test);
}
function commit(){
  @v1.aid += 1;
  @v1.arx1 += 100;
```

```
  @v1.arx2 = @v1.arx1-45;
  @v1.cid += 1;
  @v1.cx += 100;
  @v1.tid += 1;
  @v1.tx = @v1.cx;
  @v1.trsx += 100;
  var prevCommitId = @v1.currentCommitId;
  @v1.currentCommitId = getCommitId();
  ATTR("c"+(@v1.cid-1),"stroke:#888888","fill:#EEEEEE");
  AR("a"+@v1.aid,@v1.arx1,@v1.ary,@v1.arx2,@v1.ary);
  ADDGR("arrows", "a"+@v1.aid);
  CI("c"+@v1.cid,@v1.cx,@v1.cy,"","#339900","","#CCFFCC");
  ADDGR("circles", "c"+@v1.cid);
  TE("t"+@v1.tid,@v1.tx,@v1.ty,@v1.currentCommitId);
  ADDGR("commits", "t"+@v1.tid);
  FROMSYNC("a"+@v1.aid, "=", "", "scale:0", "transformOrigin:right");
  FROMSYNC("c"+@v1.cid, "=", "", "scale:0", "transformOrigin:center");
  TOSYNC("master", "=", "", "translateX:"+@v1.trsx);
  TOSYNC("head", "=", "", "translateX:"+@v1.trsx);
  if(@v1.commitIdBranches[prevCommitId]){
    TOSYNC(prevCommitId, "=", "", "translateY:"+(-80));
  }
}
function getCommitId(){
  let result = "";
  let alphabets = "abcdefghijklmnopqrstuvwxyz";
  result += alphabets[Math.floor(Math.random() * alphabets.length)]
```

```
  let characters = "abcdefghijklmnopqrstuvwxyz0123456789";
  for ( let i = 0; i < 6; i++ ) {
      result += characters[Math.floor(Math.random() * characters.length)];
   }
   return result;
}
function test() {
  alert("test");
}
function createBranch(branchName){
  if(@v1.branches.includes(branchName)){
    return alert("Branch␣"+branchName+"␣already␣exists.");
  }
  @v1.branches.push(branchName);
  if(!@v1.commitIdBranches[@v1.currentCommitId]){
    @v1.commitIdBranches[@v1.currentCommitId] = {x:@v1.tx,y:@v1.ty+80,branches
        ↪ :[]};
    GR(@v1.currentCommitId);
  }
  @v1.commitIdBranches[@v1.currentCommitId].branches.push(branchName);
  @v1.commitIdBranches[@v1.currentCommitId].y += 40;
  RETE(branchName,@v1.cx,@v1.commitIdBranches[@v1.currentCommitId].y,branchName)
      ↪ ;
  ATTR(branchName+"-rect", "fill:#FFCC66", "stroke:#CC9900");
  ADDGR(@v1.currentCommitId, branchName);
}
function deleteBranch(branchName){
```

```
    let index = @v1.branches.indexOf(branchName);
    if (index > -1) {
      @v1.branches.splice(index, 1);
      REM(branchName);
    }
  }

  $agl init = {load();}
  $git commit = {commit();}
  $git branch # = {createBranch(#);}
  $git branch -d # = {deleteBranch(#);}
```

5.3.2 Example 2: git fetch and pull

```
function pull(){
    AR("v1_a5",425,400,370,320);
    AR("v1_a6",425,400,180,400);
    CI("v1_c5",450,400,"","#888888","","#EEEEEE");
    TE("v1_t4",450,440,"7274b16");
    FROMSYNC("v1_a5", "=", "Pulling", "scale:0", "transformOrigin:right");
    FROMSYNC("v1_a6", "=", "Pulling", "scale:0", "transformOrigin:right");
    FROMSYNC("v1_c5", "=", "Pulling", "scale:0", "transformOrigin:center");
    TOSYNC("v1_master", "=", "Pulling", "translateX:300");
    TOSYNC("v1_head", "=", "Pulling", "translateX:300");
    ATTR("v1_c1","fill:#EEEEEE", "stroke:#888888");
    ATTR("v1_c2","fill:#EEEEEE", "stroke:#888888");
    ATTR("v1_c3","fill:#EEEEEE", "stroke:#888888");
```

```
  ATTR("v1_c4","fill:#EEEEEE", "stroke:#888888");
  ATTR("v1_c5","fill:#CCFFCC", "stroke:#663300");
}
function fetch(){
  AR("v1_a2",125,300,65,375);
  CI("v1_c2",150,300,"","#DDD","","#FEFEFE");
  TE("v1_t2",150,340,"7eb7654");
  FROMSYNC("v1_a2", "=", "Fetching", "scale:0", "transformOrigin:right");
  FROMSYNC("v1_c2", "=", "Fetching", "scale:0", "transformOrigin:center");
  AR("v1_a3",225,300,180,300);
  CI("v1_c3",250,300,"","#DDD","","#FEFEFE");
  TE("v1_t3",250,340,"090e2b8");
  FROMSYNC("v1_a3", "=", "Fetching", "scale:0", "transformOrigin:right");
  FROMSYNC("v1_c3", "=", "Fetching", "scale:0", "transformOrigin:center");
  AR("v1_a4",325,300,280,300);
  CI("v1_c4",350,300,"","#DDD","","#FEFEFE");
  TE("v1_t4",350,340,"ee5df4b");
  FROMSYNC("v1_a4", "=", "Fetching", "scale:0", "transformOrigin:right");
  FROMSYNC("v1_c4", "=", "Fetching", "scale:0", "transformOrigin:center");
  TOSYNC("v1_origin_master", "=", "Fetching", "x:300", "y:-230");
}
function load(){
  load_local();
  load_remote();
}
function load_local(){
  VIEW("v1");
```

```
  @v1.aid = 0;
  @v1.arx1 = 25;
  @v1.arx2 = @v1.arx1-45;
  @v1.ary = 400;
  @v1.cid = 0;
  @v1.cx = 50;
  @v1.cy = 400;
  @v1.tid = 0;
  @v1.tx = @v1.cx;
  @v1.ty = @v1.cy+40;
  @v1.trsx = 0;
  TE("v1_repository",83,50,"Local␣Repository");
  ATTR("v1_repository","font-weight:bold");
  TE("v1_current_branch",100,80,"Current␣Branch:␣master");
  load_helper(2,@v1);
  ATTR("v1_t"+(@v1.tid-1),"text:e137e9b");
  ATTR("v1_t"+(@v1.tid),"text:46d095b");
  ATTR("v1_c"+@v1.cid,"stroke:#339900","fill:#CCFFCC");
  RETE("v1_master",@v1.tx,@v1.ty+40,"master");
  RETE("v1_head",@v1.tx,@v1.ty+80,"HEAD");
  RETE("v1_origin_master",@v1.tx-100,@v1.ty+40,"origin/master");
  ATTR("v1_master-rect","fill:#FFCC66","stroke:#CC9900");
  ATTR("v1_head-rect","fill:#CCFFCC","stroke:#339900");
}
function load_remote(){
  VIEW("v2");
  @v2.aid = 0;
```

```
  @v2.arx1 = 800;
  @v2.arx2 = @v2.arx1-45;
  @v2.ary = 100;
  @v2.cid = 0;
  @v2.cx = 825;
  @v2.cy = 100;
  @v2.tid = 0;
  @v2.tx = @v2.cx;
  @v2.ty = @v2.cy+40;
  @v2.trsx = 0;
  RE("v2_background",450,250,750,10,"","","","","#EFF1FF");
  TE("v2_repository",850,25,"Remote␣Repository");
  ATTR("v2_repository","font-weight:bold");
  load_helper(4,@v2);
  ATTR("v2_t"+(@v2.tid-3),"text:e137e9b");
  ATTR("v2_t"+(@v2.tid-2),"text:7eb7654");
  ATTR("v2_t"+(@v2.tid-1),"text:090e2b8");
  ATTR("v2_t"+(@v2.tid),"text:ee5df4b");
  ATTR("v2_c"+@v2.cid,"stroke:#339900","fill:#CCFFCC");
  RETE("v2_master",@v2.tx,@v2.ty+40,"master");
  RETE("v2_head",@v2.tx,@v2.ty+80,"HEAD");
  ATTR("v2_master-rect", "fill:#FFCC66", "stroke:#CC9900");
  ATTR("v2_head-rect","fill:#CCFFCC", "stroke:#339900");
}
function load_helper(count,view){
  for(let i=0;i<count;i++){
    AR(view.name+"_a"+view.aid,view.arx1,view.ary,view.arx2,view.ary);
```

```
      CI(view.name+"_c"+view.cid,view.cx,view.cy,"","#888888","","#EEEEEE");
    TE(view.name+"_t"+view.tid,view.tx,view.ty);
    FROMSYNC(view.name+"_a"+view.aid, "=", "", "scale:0", "transformOrigin:right
        ↪ ");
      FROMSYNC(view.name+"_c"+view.cid, "=", "", "scale:0", "transformOrigin:
          ↪ center");
    if(i<count-1){
      view.aid += 1;
      view.arx1 += 100;
      view.arx2 = view.arx1-45;
      view.cid += 1;
      view.cx += 100;
      view.tid += 1;
      view.tx = view.cx;
      view.trsx += 100;
    }
  }
}
$agl init = {load();}
$git fetch = {fetch();}
$git pull = {pull();}
```

5.3.3 Example 3: adder

```
RE("",50,50,50,250);
  TE("",75,275,"PC");
  AR("",100,275,145,275);
```

```
RE("",200,110,150,250);
TE("",205,275,"Read␣address");
TE("",310,300,"Instruction");
TE("",250,350,"INSTRUCTION␣MEMORY");
AR("",350,300,395,300);
PAR("",400,225,400,480,495,480);
AR("",400,225,445,225);
AR("",400,275,445,275);
AR("",400,325,445,325);
RE("",200,210,450,200);
TE("",505,220,"Read␣register1");
TE("",605,245,"Read␣data1");
TE("",505,270,"Read␣register2");
TE("",605,295,"Read␣data2");
TE("",500,320,"Write␣register");
TE("",490,370,"Write␣data");
TE("",550,400,"REGISTERS");
LI("",550,200,550,180,"",1);
TE("",550,170,"RegWrite");
CI("",550,480,50);
TE("",550,480,"Sign␣Extend");
TE("",475,470,"16");
TE("",610,470,"32");
CI("",750,170,30);
TE("",750,160,"Shift");
TE("",750,180,"left␣2");
PAR("",600,480,695,480,695,170,715,170);
```

```
EL("",750,320,20,50);
TE("",750,300,"M");
TE("",750,320,"U");
TE("",750,340,"X");
LI("",750,270,750,230,"",1);
TE("",750,220,"ALUSrc");
AR("",650,295,725,295);
AR("",695,340,725,340);
POG("",800,220,900,240,900,320,800,340);
TE("",880,250,"Zero");
TE("",820,280,"ALU");
TE("",860,310,"ALU␣result");
AR("",650,245,795,245);
AR("",770,320,795,320);
AR("",900,250,915,250);
AR("",900,310,935,310);
PAR("",670,295,670,390,935,390);
RE("",150,140,940,270);
TE("",975,310,"Address");
TE("",1050,300,"Read␣data");
TE("",980,390,"Write␣data");
TE("",1020,350,"DATA␣MEMORY");
AR("",1090,300,1125,300);
PAR("",915,310,915,480,1110,480,1110,340,1125,340);
EL("",1150,320,20,50);
TE("",1150,300,"M");
TE("",1150,320,"U");
```

```
TE("",1150,340,"X");
LI("",1150,270,1150,250,"",1);
TE("",1150,240,"MemtoReg");
PAR("",1170,320,1190,320,1190,550,425,550,425,375,445,375);
POG("",800,80,900,100,900,180,800,200);
TE("",850,140,"Adder");
AR("",780,170,795,170);
EL("",1000,110,20,50);
TE("",1000,90,"M");
TE("",1000,110,"U");
TE("",1000,130,"X");
LI("",1000,60,1000,50,"",1);
TE("",1000,45,"PCSrc");
AR("",900,140,980,140);
POG("",200,80,300,100,300,180,200,200);
TE("",250,140,"Adder");
AR("",300,120,795,120);
PAR("",650,120,650,70,980,70);
PAR("",1020,110,1040,110,1040,30,25,30,25,275,45,275);
PAR("",120,275,120,100,195,100);
AR("",170,170,195,170);
TE("",180,160,"4");
LI("",1010,270,1010,250,"",1);
TE("",1010,240,"MemWrite");
LI("",1010,410,1010,430,"",1);
TE("",1010,440,"MemRead");
```

```
function add(){
  LI("a1",650,245,795,245,"red",3);
  FROM("a1","Read␣data1","scaleX:0","transformOrigin:left");
  LI("a2",650,295,725,295,"red",3);
  FROM("a2","Read␣data2","scaleX:0","transformOrigin:left");
  LI("a3",770,320,795,320,"red",3);
  FROM("a3","","scaleX:0","transformOrigin:left");
  LI("a4",900,310,915,310,"red",3);
  FROM("a4","","scaleX:0","transformOrigin:left");
  LI("a5",915,310,915,480,"red",3);
  FROM("a5","","scaleY:0","transformOrigin:top");
  LI("a6",915,480,1110,480,"red",3);
  FROM("a6","","scaleX:0","transformOrigin:left");
  LI("a7",1110,480,1110,340,"red",3);
  FROM("a7","","scaleY:0","transformOrigin:bottom");
  LI("a8",1110,340,1125,340,"red",3);
  FROM("a8","","scaleX:0","transformOrigin:left");
  LI("a9",1170,320,1190,320,"red",3);
  FROM("a9","","scaleX:0","transformOrigin:left");
  LI("a10",1190,320,1190,550,"red",3);
  FROM("a10","","scaleY:0","transformOrigin:top");
  LI("a11",1190,550,425,550,"red",3);
  FROM("a11","","scaleX:0","transformOrigin:right");
  LI("a12",425,550,425,375,"red",3);
  FROM("a12","","scaleY:0","transformOrigin:bottom");
  LI("a13",425,375,445,375,"red",3);
  FROM("a13","","scaleX:0","transformOrigin:left");
```

```
LI("a14",350,300,395,300,"red",3);
FROM("a14","","scaleX:0","transformOrigin:left");
LI("a15",400,300,400,225,"red",3);
FROMSYNC("a15",">","","scaleY:0","transformOrigin:bottom");
LI("a16",400,300,400,325,"red",3);
FROMSYNC("a16","<","","scaleY:0","transformOrigin:top");
LI("a17",400,225,445,225,"red",3);
FROMSYNC("a17",">","","scaleX:0","transformOrigin:left");
LI("a18",400,275,445,275,"red",3);
FROMSYNC("a18","<","","scaleX:0","transformOrigin:left");
LI("a19",400,325,445,325,"red",3);
FROMSYNC("a19","<","","scaleX:0","transformOrigin:left");
LI("a20",300,120,650,120,"red",3);
FROM("a20","","scaleX:0","transformOrigin:left");
LI("a21",650,120,650,70,"red",3);
FROM("a21","","scaleY:0","transformOrigin:bottom");
LI("a22",650,70,980,70,"red",3);
FROM("a22","","scaleX:0","transformOrigin:left");
LI("a23",1020,110,1040,110,"red",3);
FROM("a23","","scaleX:0","transformOrigin:left");
LI("a24",1040,110,1040,30,"red",3);
FROM("a24","","scaleY:0","transformOrigin:bottom");
LI("a25",1040,30,25,30,"red",3);
FROM("a25","","scaleX:0","transformOrigin:right");
LI("a26",25,30,25,275,"red",3);
FROM("a26","","scaleY:0","transformOrigin:top");
LI("a27",25,275,45,275,"red",3);
```

```
    FROM("a27","","scaleX:0","transformOrigin:left");
    LI("a28",100,275,120,275,"red",3);
    FROM("a28","","scaleX:0","transformOrigin:left");
    LI("a29",120,275,145,275,"red",3);
    FROMSYNC("a29",">","","scaleX:0","transformOrigin:left");
    LI("a30",120,275,120,100,"red",3);
    FROMSYNC("a30","<","","scaleY:0","transformOrigin:bottom");
    LI("a31",120,100,195,100,"red",3);
    FROMSYNC("a31",">","","scaleX:0","transformOrigin:left");
    LI("a32",170,170,195,170,"red",3);
    FROMSYNC("a32","<","","scaleX:0","transformOrigin:left");
  }
  $add = {add();}
```

5.3.4 Example 4: AGL architecture

```
function init(){
    RE("RE1",100,80,100,170,"","","","","white");
    TE("TE1",150,200,"High␣level");
    TE("TE2",150,220,"command");

    AR("AR1",200,210,295,210);

    RE("RE2",100,80,300,170,"","","","","white");
    TE("TE3",350,200,"Low␣level");
    TE("TE4",350,220,"AGL␣script");
```

```
  AR("AR2",400,210,495,210);

  RE("RE3",100,80,500,170,"","","","","white");
  TE("TE5",550,210,"JavaScript");

  AR("AR3",600,210,695,170);
  AR("AR4",600,210,695,250);

  RE("RE4",150,190,700,120,"","","","","white");
  RETE("TE6",775,170,"Animates␣SVG");
  RETE("TE7",775,250,"Draws␣SVG");
  TE("TE8",775,300,"UI");
}

function start(){
  on("RE1");
  LI("LI1",200,210,295,210,"red",3);
  FROMSYNC("LI1","=","Invokes","scaleX:0");
  TE("TE9",250,200,"Invokes");
  FROMSYNC("TE9","=","Invokes","scaleY:0");
  off("RE1");

  on("RE2");
  LI("LI2",400,210,495,210,"red",3);
  FROMSYNC("LI2",">","Converted","scaleX:0");
  TE("TE10",450,200,"Converted");
  FROMSYNC("TE10","<","Converted","scaleY:0");
```

```
  off("RE2");

  on("RE3");
  LI("LI3",600,210,695,170,"red",3);
  FROMSYNC("LI3",">","GSAP","scaleX:0");
  TE("TE11",650,170,"GSAP");
  FROMSYNC("TE11","<","GSAP","scaleY:0");
  off("RE3");

  on("RE4");
  LI("LI4",600,210,695,250,"red",3);
  FROMSYNC("LI4",">","SVG","scaleX:0");
  TE("TE12",650,250,"SVG");
  FROMSYNC("TE12","<","SVG","scaleY:0");
  off("RE4");
}

function on(id){
  SET(id,"fill:#CCFFCC");
}

function off(id){
  SET(id, "fill:white");
}

$agl init = { init(); }
$start = { start(); }
```

5.3.5 Example 5: duration, delay and yoyo

```
function init(){
  RE("a",100,100);
}
var angle = 45;
function rotate(){
  for(let i=0;i<5;i++){
    TO("a","","rotation:"+angle,"transformOrigin:center");
    angle += 45;
  }
}
function duration(){
  TO("a","","rotation:360","transformOrigin:center","duration:5");
}
function delay(){
  TO("a","","rotation:360","transformOrigin:center","delay:3");
}
function repeat(){
  TO("a","","rotation:360","x:300","transformOrigin:center","repeat:1");
}
function yoyo(){
  TO("a","","rotation:360","x:300","transformOrigin:center","repeat:1","yoyo:true
      ↪ ");
}
$agl init = {init();}
$rotate={rotate();}
$duration={duration();}
```

```
$delay={delay();}

$repeat={repeat();}

$yoyo={yoyo();}
```

5.3.6 Example 6: groups

```
function init(){
  RE("a");
  CI("b");
  GR("c","a","b");
  VIEW("v1");
  @v1.x = 50;
}


function start(){
  @v1.x += 50;
  TO("c","","x:"+@v1.x);
}
$start = {start();}
$agl init = {init();}
```