General Purpose Animation Language

vamsi krishna calpakkam
vamsi@iastate.edu

Follow this and additional works at: https://lib.dr.iastate.edu/creativecomponents

Part of the Graphics and Human Computer Interfaces Commons, and the Programming Languages and Compilers Commons

Recommended Citation

This Creative Component is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Creative Components by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
General Purpose Animation Language

By

Vamsi Krishna Calpakam

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
Dr. Carl K. Chang, Committee Member

Iowa State University
Ames, Iowa
2019

Copyright @ Vamsi krishna Calpakam, 2019. All rights reserved.
## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>4</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>5</td>
</tr>
<tr>
<td>NOMENCLATURE</td>
<td>6</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>7</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>8</td>
</tr>
<tr>
<td><strong>CHAPTER 1. INTRODUCTION</strong></td>
<td>9</td>
</tr>
<tr>
<td>1.1 Challenges</td>
<td>9</td>
</tr>
<tr>
<td>1.2 Organization</td>
<td>9</td>
</tr>
<tr>
<td><strong>CHAPTER 2. MOTIVATION</strong></td>
<td>10</td>
</tr>
<tr>
<td><strong>CHAPTER 3. GENERAL PURPOSE ANIMATION LANGUAGE</strong></td>
<td>11</td>
</tr>
<tr>
<td>3.1 General Idea</td>
<td>11</td>
</tr>
<tr>
<td>3.2 Architecture</td>
<td>12</td>
</tr>
<tr>
<td>3.3 Scripting</td>
<td>12</td>
</tr>
<tr>
<td>3.3.1 SVG script</td>
<td>12</td>
</tr>
<tr>
<td>3.3.2 Animation Script</td>
<td>14</td>
</tr>
<tr>
<td>3.3.2.1 Functions</td>
<td>14</td>
</tr>
<tr>
<td>3.3.2.2 Variables</td>
<td>15</td>
</tr>
<tr>
<td>3.3.2.3 Expressions</td>
<td>15</td>
</tr>
<tr>
<td>3.3.2.4 Assignment</td>
<td>15</td>
</tr>
<tr>
<td>3.3.2.5 Special Effects</td>
<td>16</td>
</tr>
<tr>
<td>3.3.2.6 Animation Sequence</td>
<td>17</td>
</tr>
<tr>
<td>3.3.2.7 Animating an entire component</td>
<td>17</td>
</tr>
<tr>
<td>3.3.2.8 Combining animation and svg script</td>
<td>17</td>
</tr>
<tr>
<td>3.3.2.9 Creating Dynamic animations</td>
<td>18</td>
</tr>
<tr>
<td>3.3.2.10 Moving SVG elements</td>
<td>19</td>
</tr>
<tr>
<td>3.4 Parser</td>
<td>21</td>
</tr>
<tr>
<td>3.4.1 SVG script parser</td>
<td>21</td>
</tr>
<tr>
<td>3.4.2 Animation Script Parser</td>
<td>21</td>
</tr>
<tr>
<td>3.5 Animation Queue</td>
<td>22</td>
</tr>
<tr>
<td>3.6 Rendering</td>
<td>23</td>
</tr>
<tr>
<td><strong>CHAPTER 4. IMPLEMENTATION</strong></td>
<td>24</td>
</tr>
<tr>
<td>4.1 IDE-Like Setup</td>
<td>25</td>
</tr>
<tr>
<td>4.1.1 Menu Panel</td>
<td>25</td>
</tr>
<tr>
<td>4.1.2 Display Panel</td>
<td>25</td>
</tr>
<tr>
<td>4.1.3 Scripting Panel</td>
<td>25</td>
</tr>
<tr>
<td>4.1.4 Template Panel</td>
<td>25</td>
</tr>
</tbody>
</table>
4.1.5 Commands Panel  25
4.2 Json Configuration File  26

CHAPTER 5. EXPERIMENTAL EVALUATION  29
5.1 Experiment 1  29
5.1.1 Results from the SVG script  30
5.1.2 Results from the animation script  31
5.1.3 Outcome  31
5.2 Experiment 2  32
5.2.1 Outcome  34
5.3 Experiment 3  35
5.3.1 Outcome  37

CHAPTER 6. RESULTS AND DISCUSSION  38
6.1 Measure of Success  38
6.1.1 Functionality  38
6.1.2 Efficiency  38
6.1.3 Modularity  38
6.1.4 Simplicity  38
6.2 Evaluation with other tools  39
6.3 Usability of GPAL  39

CHAPTER 7. RELATED WORK  41
7.1 A practical Animation Language for Software Development - Stasko  41
7.2 Visu Algo  41
7.3 Loupe  41
7.4 Git with D3  42

CHAPTER 8. CONCLUSION AND FUTURE WORK  43

REFERENCES  44
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Special Effects Example</td>
<td>16</td>
</tr>
<tr>
<td>Table 2</td>
<td>An example for Combining animation and SVG script</td>
<td>18</td>
</tr>
<tr>
<td>Table 3</td>
<td>Dynamic animations Example</td>
<td>19</td>
</tr>
<tr>
<td>Table 4</td>
<td>Example 1 for moving SVG elements.</td>
<td>20</td>
</tr>
<tr>
<td>Table 5</td>
<td>Example 2 for moving SVG elements.</td>
<td>20</td>
</tr>
<tr>
<td>Table 6</td>
<td>Example 3 for moving SVG elements.</td>
<td>20</td>
</tr>
<tr>
<td>Table 7</td>
<td>An example to illustrate timer.</td>
<td>23</td>
</tr>
<tr>
<td>Table 8</td>
<td>Scripts for Experiment 1</td>
<td>29</td>
</tr>
<tr>
<td>Table 9</td>
<td>Scripts for Experiment 2</td>
<td>33</td>
</tr>
<tr>
<td>Table 10</td>
<td>Scripts for Experiment 3</td>
<td>35</td>
</tr>
<tr>
<td>Table 11</td>
<td>Evaluation with other tools</td>
<td>39</td>
</tr>
<tr>
<td>Table 12</td>
<td>Usability of GPAL</td>
<td>40</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>GPAL Architecture</td>
<td>12</td>
</tr>
<tr>
<td>3.2</td>
<td>SVG script parser</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>Animation Script Parser</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>GPAL- IDE</td>
<td>24</td>
</tr>
<tr>
<td>4.2</td>
<td>Sample JSON configuration file</td>
<td>27</td>
</tr>
<tr>
<td>4.3</td>
<td>Workflow of JSON Config file</td>
<td>28</td>
</tr>
<tr>
<td>5.1</td>
<td>Result of SVG script from Experiment 1</td>
<td>31</td>
</tr>
<tr>
<td>5.2</td>
<td>Result of Animation script from Experiment 1</td>
<td>32</td>
</tr>
<tr>
<td>5.3</td>
<td>Output from experiment 2 - part 1</td>
<td>34</td>
</tr>
<tr>
<td>5.4</td>
<td>Output from experiment 2 - part 2</td>
<td>35</td>
</tr>
<tr>
<td>5.5</td>
<td>Result of SVG script from Experiment 3</td>
<td>36</td>
</tr>
<tr>
<td>5.6</td>
<td>Result of Animation script from Experiment 3</td>
<td>37</td>
</tr>
</tbody>
</table>
**NOMENCLATURE**

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPAL</td>
<td>General Purpose Animation Language</td>
</tr>
<tr>
<td>SVG</td>
<td>Scalar Vector Graphics</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to Dr. Simanta Mitra for his guidance throughout the course of this research and also for his support throughout my course work. I would also like to thank my co-major professor Dr. Gurpur Prabhu, who helped me with good resources and knowledge. My appreciation to Dr. Carl K. Chang for being my committee member and for his help and support.

In addition, I would also like to extend my gratitude to my parents, Mr. RaviChander, Mrs. Bhulakshmi and also my sister and brother in law for their limitless love and support throughout my career.

Last but not the least, I would like to thank my friends and colleagues who supported me and made this possible. A special thanks to my friend Ananth, for guiding me towards this successful career.

I dedicate this work to Dr. Simanta Mitra, who came up with this idea and supported me throughout the implementation.
ABSTRACT

There is a plethora of animation tools described in the computing literature. But, these tools are restricted to a particular domain or they have pre-defined algorithms that they can animate. Some examples include Git with d3, Visualgo, and Loupe etc. Most of these tools use javascript as their main background. They develop a lower level code which eventually gets converted into higher level code which is javascript. An example of this type conversion would be type script. The main goal of this project is to develop a general purpose animation tool, which can be used to animate any general animation such as moving a car, blinking a star but also, support animations that run an internal algorithm such as Git, Data Structures or animating the working of an adder in computer architecture. This project aims at developing a lower level animation script which can be extended to GUI in the future. Here, a lower level svg script is defined. The lower level script is user friendly and makes it easy for developers to adapt to it. The role of this svg script is to draw elements on the SVG element. Once these SVG elements are obtained, the main animation script can be used to animate the SVG elements. The animation script internally maintains a queue to run the animation sequentially. The animation script first goes through a parser, which internally converts the lower level code into higher level code, i.e., javascript. Some experiments were conducted using this tool, and it is seen that this tool supports most of the standard animation features and moreover easy to use, making it a powerful tool.
CHAPTER 1. INTRODUCTION

In recent years, software development has seen a tremendous growth in almost all fields such as teaching, healthcare and also technology. This is due to the power of standard languages and also an important thing is that, the standard languages are easy to learn for developers. This motivates developers to come up with their own software applications that would help others. The language developed here is powerful and uses properties of standard programming languages. And also the language is aimed to help professors create animations easily and also easily without spending much effort or learning a new language.

GPAL is aimed at simplifying animation scripts which is done using javascript. Javascript is a very difficult language to learn. But, GPAL extracts out that complexity and makes it very easy for the user to learn and use it.

1.1 Challenges

There are a lot of challenges when designing a new language. The language should meet the standards set by the already existing languages and also satisfy all the properties of a language such as the nouns and verbs of a language. And also the language should have properties such as variables, expressions, functions, loops and also conditional statements. Apart from that this language is concentrated for general purpose applications so there would be more challenges in terms of functionality.

The language is aimed at professors, so some professors might not have knowledge about animation or have coding skills, hence the language and the constructs must be super simple to use. This part is very challenging.

Apart from these another factor to be considered is the modularity of the code so that it is extensible as a standalone library in the future. This would require good coding abilities.

1.2 Organization

The report is organized as follows. First, we describe the motivation for this project, which is very interesting and the reasons for pursuing this project. Next, we discuss the language and its architecture. Third would be the relevant experiments and their results. Next, we discuss the related and similar work to GPAL. Finally some possible features for future work are discussed.
CHAPTER 2. MOTIVATION

Traditionally, people have been using powerpoint for animating certain things. It can be time consuming in some cases and people end up spending hours on it. And also creating animations with powerpoint has its own disadvantages such as the user wouldn’t be able to see the animation scripts that are being created. The users can just keep track of the animation sequence.

The existing tools are restricted for a certain use. But, however they are not interactive and follow a click and perform animation technique. This is easy for some users but, they cannot be used to create real time animations or dynamic animations.

There’s no powerful script based animation which is very interactive and also which supports features of a language such as the nouns and verbs of a language. But, having a powerful language makes it difficult for naive users or users without any programming experience. So, a language which is super simple yet powerful is necessary. Javascript, combined with SVG can act as a very powerful animation tool. But, exposing javascript for naive users is extremely challenging and would consume a lot of time learning it.

These are the motivations for working on this project with the aim of creating an animation language which is interactive, easy for naive users, supports some common nouns and verbs of a language and also which can be used for any purpose without any technical knowledge.
CHAPTER 3. GENERAL PURPOSE ANIMATION LANGUAGE

3.1 General Idea

The general idea behind General Purpose Animation Language is to make animation through scripting. The scripting is defined in a very simple low level language. A user will be able to create step by step animation sequence. The whole idea is implemented using an IDE-like setup, which allows users to create diagrams e.g., Diagram of an electronic circuit or say Git commits. These diagrams are created using the SVG script. The SVG script is converted into javascript, which parses the script and renders the diagram on the SVG. The users can also upload html code for their SVG diagrams. But, this comes with some restrictions which will be discussed in the future sections.

Later on, after the diagrams are created, they can use the animation script to animate the module that they want. The animation script uses html IDs to uniquely identify every element that is present inside the SVG. The ids of an element play a very important role in the animation part. The elements should have unique ids, otherwise there would be ambiguity in the animation.

The users are given the option of saving these SVG and animation scripts. The IDE-like setup also gives the user an option to upload an input JSON file. This JSON file has a specific format, where you can define the SVG script and animation script. These scripts will be linked to a module/function name in the JSON file. So, once the JSON file is uploaded the IDE will be able to recognize the function name and renders it automatically followed by its animation sequence.

This language also allows user to create many dynamic animations. Like creating a new commit ID in real-time or adding a node to a data structure. Apart from that, it has more capabilities such as going inside a component. For example, in electronic circuits if a user wants to view a particular component which the data is passing through they can actually do that. So, this language also supports one level deeper animation. The overall implementation is done using javascript. In the following sections we will see the architecture and how the whole scripting works.
3.2 Architecture

The general idea described in the previous section is outlined in figure 3.1

![Figure 3.1 GPAL Architecture](image)

The various components that are used in General Purpose Animation Language are:

I. Scripting  
II. Parser  
III. Animation Queue  
IV. Rendering

3.3 Scripting

Scripting, is the most important component of GPAL. It acts as an interface between the user and the animation or drawing any shapes on SVG. Scripting is the main building block for the animation and it also makes the scripting more interactive. The keywords are simple so any user can easily pick it and use them. The scripts are divided into two categories.

- SVG script  
- Animation script

3.3.1 SVG script

SVG script concentrates mostly on the drawing part. It references to drawing diagrams on the SVG. The animations will be applied on these diagrams. The syntax basically takes a keyword followed by the coordinates, attributes, and the id to be used. Some of the rules which were followed while designing the syntax are:
a) Keywords in lowercase:
   All the keywords are in lowercase. This is done to maintain uniformity.
   Eg. square

b) Keywords use appropriate shape name:
   All the keywords use the exact shape name. This is done to give more readability and
   make keywords easy to use.
   Eg. square, circle are some keywords

c) All keywords are a function:
   This means the keyword should be treated as a function. And all other things that
   follow the keywords are parameters to the function. So the parameters should be
   enclosed within open and close braces.
   Eg. square();

d) Number of parameters differs:
   The number of parameters differs for each and every keyword. Eg. The minimum
   number of parameters for a square is 4. But, it’s just 3 in case of a circle.
   Eg. square(100,150,50,50); circle(100,150,50);

e) Same keyword can have an overload on number of parameters:
   Similar to function overloading, the keywords can have an overload. (Which is the
   number of parameters)
   Eg. square(100,150,50,50); square(100,150,50,50,RID1);
triangle(x1,y1,x2,y2,x3,y3); Eg. triangle(306,292,356,442,256,442);
triangle(x1,y1,x2,y2,x3,y3,ID); Eg. triangle(306,292,356,442,256,442,RID1);

Ellipse
//The ellipse keyword takes in x and y coordinates along with x-radius and y-radius.
ellipse(x,y,x-radius,y-radius); Eg. ellipse(431,173,30,40);
ellipse(x,y,x-radius,y-radius,ID); Eg. ellipse(431,173,30,40,RID1);

Line
//The line keyword takes in x1 and y1 coordinates(starting points) along with x2 and y2 coordinates(ending points)
line(x1,y1,x2,y2); Eg. line(459,359,559,359);
line(x1,y1,x2,y2,ID); Eg. line(459,359,559,359,RID1);

Polygon
//The polygon keyword takes in n number of coordinate points.
polygon(x1,y1,x2,y2,x3,y3,x4,y4,x5,y5);
Eg.polygon(605,170,655,200,585,250,605,250,605,170);
Eg.polygon(605,170,655,200,585,250,605,250,605,170,RID1);

3.3.2 Animation Script

Animation script is used for the animation part. Animation is achieved by changing the properties/attributes of a SVG element. e.g., changing the color of an element or changing the coordinates of an element. The animation scripting is powerful, it has similarity to some existing standard languages like python. Some features of this scripting include functions, variables and expressions.

3.3.2.1 Functions

The animation scripting supports functions. The main starting point of the script is function_main, i.e., this is where the execution of the script begins. This can be considered something like a static void main function in java. So, the scripting is worthless without a function_main().

Here, define first and use later concept is used. So, if an user wants to have function calls, he has to define the function first and then make call to those functions. This idea is inspired from C, where the functions should at least be declared before we make a call to it. Functions improve code reusability.

Eg.
function_one();}
function_main(){
  one();
}

### 3.3.2.2 Variables

This scripting language supports the use of variables. The user can declare variables similar to other languages. The data types supported are integer, float, character and strings. This languages uses the power of javascript to intelligently identify the datatype of the variable. So, the user doesn’t have to worry about specifying the data type. The variables have a global scope. This can be one of the disadvantages. So, once declared the variables can be accessed throughout the program. So, we cannot reuse the variable names since they will not have local scope.

The variables are declared using the var keyword.
Eg. var(x=1000); var(y=Hello)

Multiple variables can be declared within the same statement.
Eg. var(x=10000,direction=right,branch=master,y=20000)

The values of a variable can be assigned to another variable.
Eg. var(current=x)

### 3.3.2.3 Expressions

This language supports expressions, i.e., it can perform basic mathematical operations such as addition, multiplication, subtraction and division. The expressions use javascript’s eval function. So, the eval function handles the BODMAS property of an expression. exp is the keyword for using expressions.

Eg. exp(x=x+10);

### 3.3.2.4 Assignment

Assignment is a very important property of a language. And this language supports the use of assignment operator. The values on the right hand side are assigned to the variable on the left hand side. The assignment uses the same var keyword to process the assignment functionality.

Eg. var(x=10,y=0); exp(x=x+20);
var(y=x). This would assign a value of 30 to y.
3.3.2.5 Special Effects

The main goal of this language is animation. This is achieved by changing the attributes of the SVG elements. Some attributes include the border color, border width etc. There are 4 keywords that achieve this type of animation.

Eg. on(elementID); - This one turns the border color of the element to red. This can used to highlight something while animating.

Eg. select(elementID); - This one turns the border color of the element to green. This can be used when showing important components or can be used as a normal effect apart from highlight, which is done using on keyword.

Eg. pulse(elementID); - This one creates a blinking effect. It is very useful in case of showing interactive animations such as showing the flow of current through a wire for instance.

Eg. off(elementID); - This one changes the border color of the element to black again. This can be very useful to demonstrate a state change or to indicate that some particular element has completed its animation.

All the above keywords support multiple IDS, i.e., multiple elements can be animated at the same time.

Eg. on(id1,id2,id3...);

Here is a very simple example which uses most of the important animation and svg script discussed so far.

Table 1. Special Effects Example

<table>
<thead>
<tr>
<th>SVG Script</th>
<th>Animation Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>square(376,288,150,150,sq1);</td>
<td>function_main(){</td>
</tr>
<tr>
<td>square(716,246,150,150,sq2);</td>
<td>on(sq1);</td>
</tr>
<tr>
<td>square(163,136,150,150,sq3);</td>
<td>select(sq2);</td>
</tr>
<tr>
<td></td>
<td>pulse(sq3);</td>
</tr>
<tr>
<td></td>
<td>off(s2);</td>
</tr>
<tr>
<td></td>
<td>on(s2,s3);</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>
3.3.2.6 Animation Sequence

The animations are executed step by step in the sequence specified. But, as mentioned above the users can use give effects to multiple ids at the same time. Apart from that if the user want to perform multiple type of effects on different elements then they can use the sequence keyword. You can specify any number of effects and everything will happen at the same time using the sequence keyword.

Eg. **sequence**
//Used to execute a sequence of operations in one go. Eg on something off something and pulse something at the same time.
Eg. sequence(operation1(id1,..idn)|operation2(id1,..idn)|operation3(id1,..idn));
sequence(on(RID1,RID2)|off(RID3)|pulse(RID4)|select(RID6)|on(RID7));

3.3.2.7 Animating an entire component

Another interesting feature would be combining multiple element ids into one id. In that case the users can use the names that they remember easily and can animate multiple ids at once without any burden. This is very useful when a big component is split into multiple svg ids.

Eg. **Combine**
//used to combine multiple svg ids into one single id.
Eg. combine(newSingleID,id1.id2,id3,id4,id5);
Now, the user can user newSingleID to animate id1-id5.
Eg. on(newSingleID); would turn on id1-id5.

3.3.2.8 Combining animation and svg script

The svg script is very powerful and can be used to create diagrams or shapes very easily. This is transferred over and used in the animation script to create shapes in real time. This is very useful while demonstrating an animation involving data structures for instance. So, the animation script also supports the svg script keywords and the parser is capable of distinguishing the request.

One interesting feature to note is, the user can create animation for ids which are not present but will be created in the future as the animation progresses. This is a very useful feature and stands out from other animation languages.
Table 2. An example for Combining animation and SVG script

<table>
<thead>
<tr>
<th>SVG script</th>
<th>Animation Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>square(376,288,150,150,sq1);</td>
<td>function_main(){ on(sq1); square(716,246,150,150,sq2); on(sq2); }</td>
</tr>
</tbody>
</table>

In this example above, we add an animation on(sq2). But, initially there’s no element named sq2. The language execution is run time. So, it is smart enough to map the animation to the sq2 after it is created in step2.

3.3.2.9 Creating Dynamic animations

One other important feature of this language is to create dynamic animations on the go and in real time. This would be very helpful when an user wants to add a new shape to the existing diagram during the run time. An example would be creating new nodes while demonstrating data structures. The language can draw squares/rectangles, textboxe or a circle dynamically. But, all these come up with a specific dimensions and cannot be changed. In order to have a customized dimensions the user should opt for using SVG scripts as described in 3.2.2.8.

This feature is capable of drawing the shapes in all four directions and also diagonally in all four directions. Apart from that, the user is capable of drawing arrows and connecting two shapes dynamically without specifying any dimensions or the coordinates of the shapes. But, these are only applicable to square/rectangle, text box and a circle.
Table 3. Dynamic animations Example

<table>
<thead>
<tr>
<th>SVG script</th>
<th>Animation Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>circle(198,113,50,c1); square(466,175,150,150,s1);</td>
<td>function_main(){ draw(c1,c2,circle,right); drawLeft(c1,c3,circle); drawDiagonal(s1,s2,rect,topright); drawArrow(s1,s2); }</td>
</tr>
</tbody>
</table>

The draw() keyword takes in the existing ID, the new ID for the shape that we create, the type of shape we want and the direction to draw the shape. This has the capability of getting the properties such as coordinates and dimensions of the existing shape and will render the new shape according to the direction the user specifies. The drawLeft is something similar to draw, but the direction is just specified in the keyword itself. Other functions include drawRight, drawAbove and drawBelow. Similarly the drawDiagonal() function draws the shape diagonally in the required directions. The directions include topright, topleft, bottomleft and bottomright.

The drawArrow() keyword helps the user to draw an arrow between two shapes. This is very helpful when a user wants to show that the data is being transferred between two shapes or something similar to that.

3.3.2.10 Moving SVG elements

Apart from creating dynamic shapes the users should also be capable of moving the svg elements in order to show a progressive animation. One example would be animating the head pointer in a linked list, which keeps on changing its position as new nodes are being added. So, this language has the capability of moving textbox over squares/rect, circle or another text box.
Table 4. Example 1 for moving SVG elements

<table>
<thead>
<tr>
<th>SVG script</th>
<th>Animation Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>circle(198,113,50,c1); square(466,175,150,150,s1);</td>
<td>function_main(){ draw(c1,c2,circle,right); drawBelow(c1,x,text,pointer); moveBelow(c2,x); moveAbove(s1,x); }</td>
</tr>
</tbody>
</table>

Apart from this move functionality the user can also move a svg element along its x coordinate. This is very helpful while creating general animations. One example would be moving a car. The user can combine multiple svg elements into one using the combine() function and later use move keyword to move the whole element.

Table 5. Example 2 for moving SVG elements

<table>
<thead>
<tr>
<th>SVG script</th>
<th>Animation Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>circle(198,113,50,c1); square(466,175,150,150,s1);</td>
<td>function_main(){ combine(x,c1,s1); move(x,100); move(s1,-100); }</td>
</tr>
</tbody>
</table>

This language also supports moving a SVG element slowly in order to give more life to the animation. Say suppose an user wants to demonstrate a moving of a car. Then this would be really useful. This function is implemented using threads. So, each component is treated as a separate thread and animated separately.

Table 6. Example 3 for moving SVG elements

<table>
<thead>
<tr>
<th>SVG script</th>
<th>Animation Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>rect(332,167,200,100,r1); circle(350,300,30,c1); circle(500,300,30,c2);</td>
<td>function_main(){ combine(car,r1,c1,c2); moveSlow(car,100); }</td>
</tr>
</tbody>
</table>
3.4 Parser

The next most important component of GPAL is the parser. This is where all the low level scripts are parsed into the high level javascript. The parser has multiple helper functions that helps it to identify the keywords and map them accordingly. The parser is strictly defined, meaning it expects the keywords in a certain format. Failing which, the rest of the script would fail. This property is similar to all the programming languages out there. There are two parsers used. One for the SVG script and the other for parsing the animation script.

3.4.1 SVG script parser

The SVG script parser is a powerful tool and capable of parsing the SVG script and rendering the data onto the screen. This parser uses a variety of helper functions to parse the script. Initially, it has a empty string and this empty string begins to concatenate the rendering part. After all the script is parsed this string is assigned to the SVG element to display the shapes/diagram onto the screen.

![SVG Script Parser Diagram](image)

Figure 3.2 SVG script parser

3.4.2 Animation Script Parser

This is a very important component of GPAL. This is where all the parsing happens. This parser is capable of handling animation scripts. It is powerful in the sense that it can distinguish between SVG and animation scripts. And also apart from that, it is also capable of handling the variables, and also evaluating expressions. It does everything with the help of helper functions.

This parser detects scripts that are used only for the animation and sends it to the animation queue, where it will be animated sequentially based on the time the sequence is added to the queue. Apart from the animation scripts. There are some scripts for creating variables and
performing some mathematical and assignment operations. These are carried out using the helper function and has nothing to do with the animation queue.

![Figure 3.3 Animation Script Parser](image)

3.5 Animation Queue

The animation queue makes use of javascript arrays. Javascript doesn’t have a separate queue and stack data structure. Instead javascript uses a normal array to do all the functions of a queue and stack.

So, the animation sequences are just added to the array in the same order in which they come. The methods are all synchronised so the queue is filled in the same order of request. The animation queue is initially empty and as the animation script is parsed, this queue gets filled up. This queue is very important to keep track of the animation sequence.

This language supports two ways of extracting the sequence from the queue. The first one would be very traditional way, where the user clicks a next button and the queue pops out the top element and does the animation part. The second would be through the animation script. The
user is given the power to specify the time interval for popping the element from the queue and doing the animation part. For this, the user should use the timer() keyword. The timer() keyword initializes the timer and informs the queue that it should use a timer. Next, the user should specify the time interval in which the animation should happen. There’s a keyword time() which is used to specify the time interval. It would take in one parameter and that would be the time in seconds.

Table 7. An example to illustrate timer

<table>
<thead>
<tr>
<th>SVG script</th>
<th>Animation Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>circle(198,113,50,c1);</td>
<td>function_main(){</td>
</tr>
<tr>
<td>square(466,175,150,150,s1);</td>
<td>timer();</td>
</tr>
<tr>
<td></td>
<td>time(3);</td>
</tr>
<tr>
<td></td>
<td>combine(x,c1,s1);</td>
</tr>
<tr>
<td></td>
<td>move(x,100);</td>
</tr>
<tr>
<td></td>
<td>move(s1,-100);</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

The above example would execute the animation in a time interval of 3 seconds.

3.6 Rendering

The most important component of this animation language is the rendering part. This component renders the animation onto the screen. This uses javascript to achieve that. As, explained above this language is capable of creating new SVG elements or modifying the properties or attributes of a SVG element. So, to handle two cases, the rendering component is designed in such a way to support both. The ID of a SVG element is very important if it is changing the attribute of that element. Without the ID, second component would become worthless. So, it is always a good practise to create SVG elements with the IDs.
CHAPTER 4. IMPLEMENTATION

The whole implementation is in javascript. The code is separated into parser, helper and rendering part for better maintainability and extensibility. For the user to create animation and test them, an IDE-like setup is created. This IDE-like setup allows the users to create SVG scripts and also animation scripts. The users can also test their scripts before demoing it. This is a very user friendly IDE and users can use it very easily without any prior experience. Everything is very straight forward. In the upcoming section we will go through the important components of the IDE.

4.1 IDE-Like Setup

There’s an IDE-like setup that allows user to input their scripts. There are few components which are a part of the IDE such as the menu panel, the template panel, the display panel, the scripting panel and the commands panel.

Figure 4.1 GPAL- IDE
4.1.1 Menu Panel

This is simple menu bar as in all applications. It has certain functions such as opening a JSON file or opening an existing SVG code. Some other functionalities include a link to the documentation page and an evaluate button which evaluates the SVG script and also a animate button to evaluate the animation script.

4.1.2 Display Panel

This is the main display panel that is made up of SVG element. And all the diagrams come under this SVG. Whatever changes that are made are shown in this panel and also the animation takes place in this panel. This can be considered as the main viewing panel.

4.1.3 Scripting Panel

Scripting panel is where all the scripting is done. There are two tabs, one for SVG scripting and one for animation scripting. The user can switch between the tabs and write their scripting accordingly.

4.1.4 Template Panel

Template panel is similar to a drag and drop in a general SVG editor. It has a certain predefined shapes that the user can just select the shape they want and click anywhere on the display panel to draw that shape. This not only draws the shape but also generates equivalent SVG script, which can be seen on the scripting panel. Also, one important point to note is that, everytime the user draws a shape using this functionality, a random ID is created for that shape. Which can be very useful while creating the animations.

4.1.5 Commands Panel

Commands panel provides a way for real-time animation based on the commands entered. But, if an user wants to use this functionality. The user should specify the SVG/animation script in the configuration JSON file(which will be discussed in the upcoming section.). Once the user uploads this configuration file. The user can simulate animations by just entering the commands. e.g., A user can simulate Git commit by entering their commands in this panel.
4.2 Json Configuration File

A configuration file in the form of JSON is used for many reasons, one important reason would be reusability of scripts. And also as discussed above if the user wants to use the commands panel, then the IDE would expect the JSON file.

The JSON configuration file supports various features such as module name, and the corresponding SVG and animation scripts for that module. And also one very important feature is that, with the configuration file the users can make a function accept parameters which is not possible with this. So, once the user defines the function name along with the SVG and animation scripts in the configuration file, then the user can simply reuse those functions in the animation script that they write or call those functions using the commands panel. One other important feature would be renaming the SVG elements to the name that the user can easily remember. For example, the user can rename a SVG element to whatever name he prefers and this can be done easily through the JSON file. A simple configuration file is shown below.

Mappings is very useful when the user can’t keep track of SVG element ids. The user can simply change the SVG ids to the name they want. The language intelligently distinguishes this and maps the new name to the old name during runtime. The old name would go inside the source value and the new name would go under destination. The mappings is a JSON array, so the user can have any number of mappings.

Next would be the functions, which is also a JSON array and hence can have multiple functions defined here. Some properties of a function would be name, sequence, svg and the parameter.

Name: This would define the name of the function. And this should stay unique throughout the execution. It is not capable of handling overloads.

Parameter: This would say if the function would accept parameters or not. So, the values would be either yes or no. One interesting thing is that when this is set to yes it is not compulsory to send values. The language is capable of using the old values. This is because the variables have global scope.

Sequence: This defines the sequence of animation. One thing to note is that all the sequence do not have a ‘;’ at the end. The language appends a ‘;’ at the end of the sequence. And also, it expects each sequence to be separated as shown in the example. It wouldn’t accept two or three sequence appended together.

Svg: This defines the SVG script associated with the function. Similar to sequence it doesn’t accept multiple sequences together. So, use only one sequence as a JSON value in the JSON array.
Similarly, the user can create animation and SVG scripts using the IDE and transfer it the JSON file. The functions array can support any number of functions. The workflow of from creating the configuration file and using the function through the commands panel is shown in the diagram below.
Figure 4.3 Workflow of JSON Config file

The first step would be creating the JSON file and the user can upload the JSON file through the IDE. Then the languages parses the JSON file and stores the animation and SVG sequence using the javascript data structures. Now, the user can use the commands panel and enter the commands they want.

Eg. Say suppose the Json file has a function name commit and its corresponding animation and SVG sequence then the user can simply type commit(); on the commands panel.
CHAPTER 5. EXPERIMENTAL EVALUATION

Initially, this language was targeted and restricted to computer architecture animations. So, the basic experiments were related to computer architecture. The majority part of the experiments include animating the multicycle datapath in computer architecture. The language is capable of animating many use cases such as GIT, data structure, or any general animations such as moving of a car and many more. All the experiments that were tried using GPAL are available in a common place under home.html

5.1 Experiment 1

DataPath animation for Rtype add.

For this experiment, the main essential things are the SVG script and the animation script. And interesting thing to note is that, the diagram is very complicated but it was drawn using GPAL’s SVG script. And the path was also defined using the animation script. The sample scripts used for this experiment are shown below.

Table 8. Scripts for Experiment 1

<table>
<thead>
<tr>
<th>SVG script</th>
<th>Animation Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>pc(42,271,programCounter); im(131,238,instructionMemory); register(430,200,register); mux(370,307,mux1); mux(665,280,mux2); mux(930,80,mux3); mux(1042,300,mux4); alu(744,175,alu); dm(880,250,dataMemory); elliptical(498,412,Sign Extend,signExtend); elliptical(712,426,ALU Control,aluControl); elliptical(663,130,Shift left 2,shift); adder(144,40,adder1); adder(755,55,adder2); cl(280,210,430,210,Instruction[25-31],rs); cl(280,250,430,250,Instruction[20-16],rt); cl(280,411,420,411,Instruction[15-0],imm16); cl(280,320,350,320,Instruction[15-11],rd); cl(78,298,130,298,,pcToM); cl(390,296,430,296,,mux1ToReg); cl(420,411,470,411,16,line16); cl(530,411,620,411,32,line32); pcl(1061,318,1088,318,1088,529,406,529,406,33,430,333,mux4ToReg); pcl(948,67,960,67,960,17,17,17,17,298,42,298,P CtoMux3); pcl(420,411,420,471,670,471,670,430,683,430,16toALUcontrol); pcl(742,430,769,430,769,286,aluControlToALU); pcl(828,259,829,375,1000,375,1000,310,1025,31 3,430,333,mux4ToReg); pcl(948,67,960,67,960,17,17,17,17,298,42,298,P CtoMux3); pcl(420,411,420,471,670,471,670,430,683,430,16toALUcontrol); pcl(742,430,769,430,769,286,aluControlToALU); pcl(828,259,829,375,1000,375,1000,310,1025,31</td>
<td></td>
</tr>
<tr>
<td>function_state0(){ }</td>
<td>function_state0(){ }</td>
</tr>
<tr>
<td>function_state1(){ }</td>
<td>function_state1(){ }</td>
</tr>
<tr>
<td>function_state6(){ }</td>
<td>function_state6(){ }</td>
</tr>
<tr>
<td>on(RegisterToAlu);</td>
<td>on(RegisterToAlu);</td>
</tr>
<tr>
<td>on(alusrc,RegisterToMux2,Mux2ToAlu);</td>
<td>on(alusrc,RegisterToMux2,Mux2ToAlu);</td>
</tr>
<tr>
<td>on(aluControlToAlU);</td>
<td>on(aluControlToAlU);</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>function_state7(){ }</td>
<td>function_state7(){ }</td>
</tr>
<tr>
<td>on(regwrite);</td>
<td>on(regwrite);</td>
</tr>
<tr>
<td>on(aluToMux4,mux4ToReg);</td>
<td>on(aluToMux4,mux4ToReg);</td>
</tr>
<tr>
<td>on(regds,rd,mux1ToReg);</td>
<td>on(regds,rd,mux1ToReg);</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>function_final(){ }</td>
<td>function_final(){ }</td>
</tr>
<tr>
<td>on(pcToAdder1,adder1ToMux3,PCtoMux3,next4));</td>
<td>on(pcToAdder1,adder1ToMux3,PCtoMux3,next4));</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>function_main(){ }</td>
<td>function_main(){ }</td>
</tr>
<tr>
<td>state0();</td>
<td>state0();</td>
</tr>
<tr>
<td>state1();</td>
<td>state1();</td>
</tr>
<tr>
<td>state6();</td>
<td>state6();</td>
</tr>
<tr>
<td>state7();</td>
<td>state7();</td>
</tr>
<tr>
<td>final();</td>
<td>final();</td>
</tr>
</tbody>
</table>
5.1.1 Results from the SVG script

When the above SVG script is executed, the equivalent SVG diagram would be generated as seen below. There’s nothing hard coded in this diagram everything goes through the SVG script to generate the diagram.
5.1.2 Results from the animation script

On running the animation script the animation would take place sequentially when the user manually presses the next button. This is very interesting. Because there’s no other tool that can make it this interactive to create the diagrams and also animate them through scripting.

5.1.3 Outcome

This visual datapath animation was shown to students of COM S 321 class and were shown Rtype addition, branching and load word. This animation is believed to have improved the student’s grade. It was noticed that the students were more involved as they were shown something visually and something more realistic. So, the visual animation might have been one of the factors for improving their grade. And also, apart from that the scripts were pretty easy to create and consumed around 3 hours to create the SVG and animation script. But, as the language has progressed now it would take around 1-2 hours to create this type of animation. One advantage is that, these are permanent so the time spent is only once and can be used for many years.
5.2 Experiment 2

Animating Git.

The next challenge was to animate GIT, which is a very general and complicated animation. But, the language was able to do that very effectively and also easily. But, there’s some pros and cons as well. The language is not capable of detecting the exact positions of an element. So, the user will have to feed those data manually while creating say new commit objects. Apart from that the animation works smoothly. One other disadvantage is that, this language couldn’t animate the very complicated cases of git pull and push which includes a remote and local repository. The GIT animation was illustrated using the configuration file and using the commands panel. Below is the configuration file used for GIT animation.

After uploading this configuration file, the user will have to change to animation mode. Then, the user can start using the commands panel and start with init(); command, which is an equivalent to git init. Below, we can see the init(); command being executed. The user should manually press the next button to view the sequence of animation.
Table 9. Scripts for Experiment 2

<table>
<thead>
<tr>
<th>Script Name</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
<td><code>var(x=10000,direction=right,branchName=master,y=20000)</code>, <code>circle(350,275,50,x)</code>, <code>var(current=x)</code>, <code>drawBelow(x,y,text,x)</code>, <code>drawBelow(y,branchName,text,master)</code>, <code>drawBelow(branchName,head,text,head)</code>, <code>exp(x=x+10)</code>, <code>exp(y=y+10)</code></td>
</tr>
<tr>
<td><strong>commit</strong></td>
<td><code>draw(current,x,circle,direction)</code>, <code>drawBelow(x,y,text,x)</code>, <code>drawArrow(x,current)</code>, <code>var(current=x)</code>, <code>moveBelow(y,branchName)</code></td>
</tr>
<tr>
<td><strong>branch</strong></td>
<td><code>drawBelow(head,name,text,name)</code></td>
</tr>
<tr>
<td><strong>checkout</strong></td>
<td><code>moveBelow(name,head)</code>, <code>var(branchName=name)</code>, <code>pointer(current=branchName)</code></td>
</tr>
</tbody>
</table>

**Git.json**

```json
{
    "mappings": [],
    "functions": [
        {
            "name": "init",
            "sequence": [
                "var(x=10000,direction=right,branchName=master,y=20000)",
                "circle(350,275,50,x)",
                "var(current=x)",
                "drawBelow(x,y,text,x)",
                "drawBelow(y,branchName,text,master)",
                "drawBelow(branchName,head,text,head)",
                "exp(x=x+10)",
                "exp(y=y+10)"
            ]
        },
        {
            "name": "commit",
            "sequence": [
                "draw(current,x,circle,direction)",
                "drawBelow(x,y,text,x)",
                "drawArrow(x,current)",
                "var(current=x)",
                "moveBelow(y,branchName)"
            ],
            "parameter": "yes"
        },
        {
            "name": "branch",
            "sequence": [
                "drawBelow(head,name,text,name)"
            ],
            "parameter": "yes"
        },
        {
            "name": "checkout",
            "sequence": [
                "moveBelow(name,head)",
                "var(branchName=name)",
                "pointer(current=branchName)"
            ],
            "parameter": "yes"
        }
    ]
}
```
After init();, the next command would be committing and creating a new commit object. This is just an example, the user can do whatever they have defined in the configuration file. Here, commit is being executed. The command would be commit(); These names refer to the name you give in the configuration file.

**5.2.1 Outcome**

An existing tool for animating GIT would be git with d3. But, this tool is capable of animating only GIT and hence is not extensible to other applications. Here, the GPAL is capable of animating GIT smoothly similar to git with d3. But, both has it’s own pros and cons. GPAL is not capable of animating the remote and local repository, i.e., there’s no way of dividing the svg into several components. But, as the language progresses there would be a solution for this.
5.3 Experiment 3

It’s always better to show something visually. So, the final experiment would be to demonstrate the working of GPAL. An important thing to note is that it took 15 minutes to create both SVG and animation script to demonstrate the working of GPAL. Below is the script which is used for creating this animation.

Table 10. Scripts for Experiment 3

<table>
<thead>
<tr>
<th>SVG script</th>
<th>Animation Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>square(50,160,100,100,s1);</td>
<td>function_main()</td>
</tr>
<tr>
<td>square(50,360,100,100,s2);</td>
<td>{</td>
</tr>
<tr>
<td>circle(550,256,70,c1);</td>
<td>on(s1);</td>
</tr>
<tr>
<td>text(50,136,lowLeveLCode,t1);</td>
<td>drawArrow(s1,s2);</td>
</tr>
<tr>
<td>text(50,483,Parser,t2);</td>
<td>pulse(s2);</td>
</tr>
<tr>
<td>text(480,360,JavaScript,t3);</td>
<td>on(s2);</td>
</tr>
</tbody>
</table>
Below, we can see the results from the SVG and animation script. Please run it manually to understand the working of GPAL.

Figure 5.5 Result of SVG script from Experiment 3
5.3.1 Outcome

This animation was created in a very short span of time, showing that GPAL is also a powerful language with some cons. But, as there’s incremental growth the language would be even more powerful.
CHAPTER 6. RESULTS AND DISCUSSION

6.1 Measure of Success

6.1.1 Functionality

As seen from the above examples, it is obvious that GPAL is providing various functionalities. These functionalities include creating diagrams using SVG and animating the SVG elements. The functionalities are more concentrated towards the animation part of it. But, there can be a lot more features that can be added to this language. Those will be discussed in the future work section. This would be the starting point, where developers can include or extend the features.

6.1.2 Efficiency

The language is lightweight and is efficient in converting the scripts to javascript, so the user no longer has to worry about the runtime or writing complicated scripts. Everything would essentially run in linear time unless the users opt to use loops and make it complicated. Otherwise, the language is super efficient and light.

6.1.3 Modularity

The code is modularised into standalone functions. The main motive was to separate out the language as separate stand alone library. The code is modularized into parser, helper and renderer. So, the code is maintainable and the developers have to make very little changes to implement new functions. Modularity is highly helpful. The experiments web page uses the modularized code. So, various inputs were fed into modularized code and the output appears on the screen. So, it is seen that the language is quite powerful and capable of extending into a standalone library.

6.1.4 Simplicity

The main goal of this language is simplicity and makes it easy for novices or users very new or with no knowledge about programming. They should be able to use this language without any prior programming language. This is achieved by giving very basic and easy names to the function and moreover the language structure is pretty good to learn and adapt to it easily. This language accomplices the simplicity goal.
6.2 Evaluation with other tools

GPAL is inspired by many existing animation languages out there such as visual algo, git with d3, loupe and many more. So, GPAL is evaluated with those languages on what it can and what it cannot do. This is shown in the table below. It is seen that GPAL is capable of animating almost all uses cases.

Table 11 Evaluation with other tools

<table>
<thead>
<tr>
<th></th>
<th>Visual Algo</th>
<th>Git with D3</th>
<th>GPAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>321</td>
<td>NO</td>
<td>NO</td>
<td><strong>YES</strong>, Multi cycle data path is the main focus.</td>
</tr>
<tr>
<td>311</td>
<td><strong>YES</strong>, Algorithms and data structures are the main focus.</td>
<td>NO</td>
<td><strong>YES</strong>, But the user should give the sequence of the animation.</td>
</tr>
<tr>
<td>GIT</td>
<td>NO</td>
<td><strong>YES</strong>, that's the main functionality and it is pretty smooth.</td>
<td><strong>YES</strong>, but the user should draw the diagram manually and keep track of the pointers manually.</td>
</tr>
<tr>
<td>State Transitions</td>
<td>NO</td>
<td>NO</td>
<td><strong>YES</strong>, the user can turn on and turn off the states and show visual transformation</td>
</tr>
<tr>
<td>General Animations</td>
<td><strong>No, it is restricted.</strong></td>
<td><strong>No, it is restricted.</strong></td>
<td><strong>Yes, for example we can simulate moving of a car.</strong></td>
</tr>
</tbody>
</table>

6.3 Usability of GPAL

Other than the features, the goal of the language is to be simple and usable by even naive users. The language satisfies most of the conditions yet there are some things that should be improvised in the future work. The features of this language are evaluated based on how a naive user would use this tool with a user with coding knowledge and it is tabulated below.
Table 12. Usability of GPAL

<table>
<thead>
<tr>
<th>Core Features</th>
<th>Animation Language - Naive User</th>
<th>Animation Language - User with coding knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Draw a diagram</strong></td>
<td>Can use drag and drop.</td>
<td>People with coding knowledge can directly use the script to draw a diagram.</td>
</tr>
<tr>
<td><strong>Paste a diagram from external source</strong></td>
<td>Yes, The only accepted form is <strong>SVG</strong></td>
<td>Yes.</td>
</tr>
<tr>
<td><strong>Change the position of the image</strong></td>
<td>Once the shape is drawn using drag and drop, the position can be changed only through the code.</td>
<td>Yes, the user can change the position using the script. But, should be aware of the syntax and the coordinates system.</td>
</tr>
<tr>
<td><strong>Animate the shapes</strong></td>
<td>No. They need basic coding knowledge to use the keywords to animate</td>
<td>Yes. The keywords and the coding logics are inspired from javascript. So, the can use it once they know the syntax.</td>
</tr>
<tr>
<td><strong>Move the shapes during animation</strong></td>
<td>No. Coding and coordinates knowledge is required.</td>
<td>Yes. But good amount of coding and coordinates knowledge is required.</td>
</tr>
</tbody>
</table>
CHAPTER 7. RELATED WORK

There’s a lot of related and similar work. That’s because there’s been a tremendous development in the field of animations ranging from basic 2D to the latest AR based animations. Here, we discuss about some of the related work.

7.1 A practical Animation Language for Software Development - Stasko

Stasko would be the starting point of animation languages. Where they developed a very basic animation language to support 2D animations. Some interesting features include.

- Uses 4 different types to identify an object.
- Every element is considered as a separate object.
- Location: Specifies the exact position of object say x and y coordinate
- Images: Specifies the type of images. They have considered the basic shapes such as square, rect and circle.
- Images are of two types… One is the basic images comprising of basic shapes.
- Second is a composite image having all two or more basic images. For example, consider a door it has rect and a circle.
- Paths: Specifies the movement of image in the particular direction, e.g., path of length 3. Has x1 y1 x2 y2 and x3 y3. So, the image moves to these coordinates at some time.
- Transition: Has three parameter. One is the trans type, two is the image and three being the path.
- Trans types used here are move, resize, fill, color, visibility.

7.2 Visualgo

Visu algo is a web based tool for animating algorithms. This tool is really powerful and can animate a certain set of animations. Interesting this is that this tool allows users to create their own examples and run the algorithms on them. But, the main disadvantage is that, this tool cannot create any new animations so there’s no way for the user to come up with scripts to animate a new algorithm.

7.3 Loupe

Loupe is a web based animation tool to animate the working of javascript. This is also a very powerful tool and works very smoothly. But, again this is restricted to the working of javascript
and the user will not be able to animate whatever they like. So, that would be a drawback of this tool. But, the working of this tool is one of the inspiration for GPAL. This tool uses an external parser which in turn uses helper functions and finally execute the animation. The exact workflow is followed in GPAL.

7.4 Git with D3

Git with d3 is tool to animate the working of git. This tool has attracted many people and students. It uses D3 for the animation part. And the functions are hardcoded to do specific task. So, it is not entirely functional i.e., it has many bugs. But, it is very good and an innovative tool. This tool gives the idea of having two panels for animation, which is not supported in GPAL. Here, they animate local and remote repos using two panels.
In this project, a language is created to support creation of SVG elements and animating those elements. The language is focused on general applications so the scope of the language is not restricted. It is also evident that the language is simple to use so, any user can use this language without any prior programming knowledge. But, the language doesn’t have many properties of a general programming language. So, there can be a lot of improvements in the future.

The language can be extended to support local variables and also global variables. Right now it only supports global variables. Other improvements can be made to the properties of a function. Such as, making the language to support a return type. The language can also be extended to support if statements. In order to do that, the language must support conditions. Right now, the language can support expressions but not conditions. This can be easily implemented by using javascript conditions. Apart from that the language can be easily converted to support object oriented properties, in that case the language itself would be able to operate as a standalone library and can be used a powerful animation engine.

Since the language focuses more on the animation part, it would be great if more animation oriented functions come up such as rotate, transform, or something like a bounce. And also, some interesting features such as defining two or more areas for animation. One particular use case for this feature would be git local and remote repository animation.

This language can be extended with multiple interesting features such as 3D animations or animations involving AR to make things even more visual and in the future can be a very powerful animation language.
REFERENCES


