AN INTERACTIVE INTERFACE FOR SHADER PROGRAMMING

by

Izza Tariq

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science

Department of Computer Graphics Technology

West Lafayette, Indiana

May 2020

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Esteban Garcia, Chair
Department of Computer Graphics Technology

Dr. Tim McGraw
Department of Computer Graphics Technology

Andres Colubri
Processing Foundation

Approved by:
Nicoletta Adamo

Dedicated to my parents

ACKNOWLEDGEMENTS

I would like to thank my academic advisor Dr. Esteban Garcia for his constant support, guidance, and encouragement throughout this research project. Thank you to my other two committee members, Dr. Tim McGraw and Andres Colubri for being great mentors. I would also like to thank the Processing Foundation for assisting in this study and giving me the opportunity to contribute to the promotion of software literacy within visual arts using their open-source platform.

My deepest gratitude and love to my parents and siblings who were always by my side despite being thousands of miles away from me. Your warmth, love and constant words of encouragement lifted me in times when I needed it most.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Shader programming is an efficient way to render graphics yet can be complex to understand for emerging computational artists who are new to the world of creative programming. Since shaders are an important part of visual programming, making them easier to use should be considered an important factor in teaching basic to advanced concepts in the rendering pipeline of computer graphics. Though some tools like shader editors and interactive graphical interfaces have been designed to aid in the understanding of these concepts, they still fall behind in some areas in meeting the requirements of emerging computational artists.

In this thesis we present and explore the usability of a new Shader Mode tool developed for an open-source software called Processing, widely used in education and production for digital arts and design, and discuss how it can aid shader programming for emerging computational artists. This tool was inspired by a project the author did for Google Summer of Code in 2018. The tool was tested with Processing users who have some familiarity with shader programming. The analysis of the tool was done using a mixed method research technique: combination of quantitative and qualitative analysis. Since we were unable to get enough users for the study, the results could not be generalized to all Processing shader users. However, relevant user feedback is discussed.

CHAPTER 1 INTRODUCTION

1.1 Motivation

This study was inspired by a project the author did for Google Summer of Code in summer 2018 (GSOC'18) and was later continued as part of the author's master's thesis. The project dealt with creating a prototype for a shader editor tool for the Processing Development Environment. The motivation behind creating this tool was to extend the shader programming feature of Processing. Increasing numbers of computational artists and designers have started using Processing to create their artwork. Shader programming is a relatively new feature in Processing that is expected to extend the possibilities of real-time rendering for these artists. At the same time, it is challenging for artists and designers who come from a non-technical background with little to no programming experience.

The initial goal was to provide Processing users with a text-based shader editor that opened from within the PDE and allowed real-time editing of shader code, a process known as live coding. Though the tool developed in GSOC met our initial basic requirements, it did not provide a smooth workflow and often led to inconsistent rendering displays because of conflicting naming conventions of default variables. Moreover, there was room for more features that could enhance the experience of editing shaders even more. Therefore, it was decided to take the project forward as part of an implementation with a user-centered design study.

1.2 Significance

Shader programming can be very useful for computational artists as it increases the possibilities for them in visual programming and real-time rendering (Gomez, 2016). A number of popular computational artists such as Joshua Davis (Davis, n.d.), Abe Pazos (Pazos, n.d.) and Raven Kwok (Kwok, n.d.) have used shader programming to create stunning pieces of visual art which shows great potential of shaders as a creative technique in computational design. However, most of these artists come from a non-technical background and struggle with learning this technique as setting up shaders alone can be quite cumbersome (Owen, 2005). Moreover, lack of technical knowledge makes it harder for them to keep up with advancements made in GPU hardware that can directly affect the use of graphics APIs.

The purpose of this study was to explore if the introduction of a new Shader Mode in Processing could help bridge the gap between the creative ability of computational artists and the advanced technical expertise required for shader programming. Bridging this gap can help make the process of learning and programming shaders easier for emerging computational artists and open new creative directions for them in their work.

1.3 Scope

This research dealt with developing an interactive shader interface for Processing users familiar with shader programming. The tool features a text-based editor that supports both PDE files (Processing sketch file type) as well as GLSL files. The interface was built as a tool for the Processing Development Environment (PDE), therefore it is only accessible and functional from within the PDE. Participants for our study were chosen from within an online Processing community who were at least eighteen years old and had some experience with shader programming. Since the study was conducted online, these users could be from anywhere around the world.

The goal of the research was to develop the tool and test how usable it was for our targeted user group. Users were given a fixed time frame to experiment with the tool using their own PCs and create something of their choice using shader programming. Evaluation of our study was done using a combination of quantitative and qualitative analysis also known as mixed methods research. We chose this method to get a deeper understanding of the quantitative results using qualitative data and clarify anomalies if any in the former (FoodRisc Resource Centre, n.d.). Our quantitative data was analyzed using descriptive and inferential statistics while the qualitative data was analyzed using an approach called open coding.

1.4 Research Questions

This study focused on exploring the usability of a new Shader Mode inside Processing. Our research questions were as follows:

1. Does Shader Mode have a high usability for all Processing users familiar with shader programming?
2. Is the usability of Shader Mode dependent on expertise of users in shader programming?

1.5 Variables

Following are the variables for this study.

- Independent Variables
    - Shader Mode tool

- Dependent Variables
    - Usability

- Nominal Variables
    - Expertise in shader programming

1.6 Assumptions

This research was based on a few assumptions. We assumed that the following factors did not have a significant effect on the usability of Shader Mode:

- The type of framework, which in our case is the Processing framework
- Self-reported expertise is accurate
- The amount of time spent to complete the task

1.7 Limitations

Due to the current disruption in world activity that resulted from COVID-19, we got limited participants for our study (less than 10 users). Even though 30 users signed up for the study, only 8 users reported their feedback. Because of this, our results are not generalizable to the entire Processing community who uses shaders.

Because of very few users signing up for the study, we did not include a control group that was supposed to perform the experiment using the traditional way i.e. using the Java Mode in Processing Development. Therefore, we cannot compare the usability of the Shader Mode with that of the Java Mode in regard to programming shaders in Processing.

After reviewing the literature, we found no prior experiments that conducted usability tests on this domain (shader writing). Therefore, we will not be able to compare our results with the results of other studies and hence would not be able to make a clear statement on the usability of our tool as compared to other shader editor tools.

1.8 Delimitations

Our study will involve users from within the Processing community. These users will be further narrowed down to those who have some experience in shader writing and have a background in computational arts. The testing of the tool will be done remotely. Users will download the tool and test it on their personal machines.

There are a number of shader types that can be used to render a scene. These include: vertex, fragment, geometry and tessellation. However, our shader editor tool will only include the first two. This will limit the range of shader effects that could be applied to render a scene, thus limiting the possibilities of shader development for users.

1.9 Definitions

Shader – A shader is "a custom shading and lighting procedure that allows the motivated artist or programmer to specify the rendering of a vertex or pixel" (Fosner, 2003)

High-level programming – The term "higher level" means that many details are handled automatically so that programmers can write less code to get the same job done (Ousterhout, 1998)

Low-level programming – Each statement represents a single machine instruction and programmers must deal with low-level details such as register allocation and procedure calling sequences (Ousterhout, 1998).

Real-time rendering – animations that are rendered so quickly they appear to be being generated in absolute real time. (Vivo & Lowe, n.d.)

CHAPTER 2 LITERATURE REVIEW

2.1 Introduction

A shader is “a custom shading and lighting procedure that allows the motivated artist or programmer to specify the rendering of a vertex or pixel” (Fosner, 2003). It is a set of instructions that are executed all at once for every single pixel on the screen or for every vertex of a model (Vivo & Lowe, n.d.). Shaders follow the rules of *parallel processing* in which many calculations or processes take place simultaneously. This mechanism reduces the overhead on CPUs thus decreasing processing time and producing faster rendering results.

2.2 History

Before the year 1989, the fixed function pipeline in the graphics processing unit (GPU) allowed only limited changes to be made to the final computer-generated imagery like pixel transformation or shading. This limited the number of ways to achieve a desired look. However, with the evolution of GPUs, this fixed pipeline became programmable through the use/introduction of shaders. One of the earliest shaders were developed and introduced by Pixar in their rendering software Renderman as part of their Renderman Specification Interface (RISpec) API (Apocada, 1990). It consisted of Renderman Shading Language (RSL), a C-like programming language which could be used to develop four types of shaders: surface, displacement, light and volume shaders. This was the major component used in achieving the marvelous cinematic effects in their famous movie Toy Story (Maughan, 2001). Though many of these effects were achieved through shaders, they were not real-time: it took months to render them on powerful workstations.

The desire to replicate those effects resulted in GPU hardware companies making their graphics processing units more powerful to allow more and more of the rendering pipeline to be programmable. This allowed them to create different visual effects on the fly i.e. while the program is running. In February 2001, NVidia introduced the first ever graphics card chip, known as GeForce 3, capable of programming shaders (Luebke, 2007). The era of real-time rendering had begun. Following that, major graphics software libraries began developing shading languages to program the new programmable GPU rendering pipeline: OpenGL group developed OpenGL Shading Language (GLSL) (OpenGL Shading Language) while Direct3D developed High-Level Shading Language (HLSL) (Satran, n.d.). Through the use of these languages, special algorithms that allowed the manipulation of color, hue, saturation, geometry, vertices and textures of pixels in real-time could be defined in the shader. These parameters could also be modified by external parameters declared outside the shader. This flexibility allowed for a wide range of possibilities for special effects in real-time rendering. This could in turn give greater freedom to visual artists and programmers to create desired visual effects for their application (Fosner, 2003).

2.3 Processing

Despite the essential part that shaders play in the field of computational arts, their popularity amongst designers and artists is still very low (Gomez, 2016). A major reason for this is the need for advanced programming knowledge which keeps increasing as graphics APIs and hardware evolve rapidly (Gomez, 2016).

Computational artists and designers are at a disadvantage when this happens as they do not have extensive programming background. Since their field heavily depends on interactive computer graphics, some open-source frameworks have been developed specifically for such people to facilitate the teaching of relevant programming skills they require to build interactive applications. The most prominent amongst these is Processing, a minimal integrated development environment initiated by Casey Raes and Ben Fry that uses a java-based programming language (Processing Foundation, n.d.).

The reason Processing stands out from other frameworks is that it is specifically geared towards computational artists and designers and has a strong focus on “increasing computer literacy within the design and visual arts, and visual literacy within technology and engineering” (Reas, 2014). The simple API and interface helps beginners progress gradually from basic to complex shader programming. Processing does not only have a vast drawing API, comprising around 300 functions, but it also supports the use of multiple other graphical libraries most of which have been contributed by the open-source community. On September 30, 2015, Processing also released their complete shader API compatible with the Processing API thus enabling real-time rendering and giving rise to more exciting possibilities in interactive visualizations. Hence, we chose the Processing framework for our study.

2.4 Real-time Shader Programming

Amongst the few shader prototyping interfaces developed, RenderMonkey has been the most popular one (RenderMonkey Toolsuite, n.d.). However, it is deprecated now as the company that produced it no longer supports it. One of the key unique characteristics of this tool that sets it apart from other tools is that it has been created and designed in a way that meets the requirements of both programmers and artists (Tatarchuk, 2004). This was achieved by having two separate interfaces, a text-based code editor for programmers (Figure 2.1) and a visual editor for artists (Figure 2.2) with no experience in programming. Each effect of a shader can be visualized in a tree structure in a separate panel. This helps in examining the gradual build-up of effects in the rendering window. The customizable user interface with controllable features such as font size, keyword colors and tab settings offer a user-friendly environment.

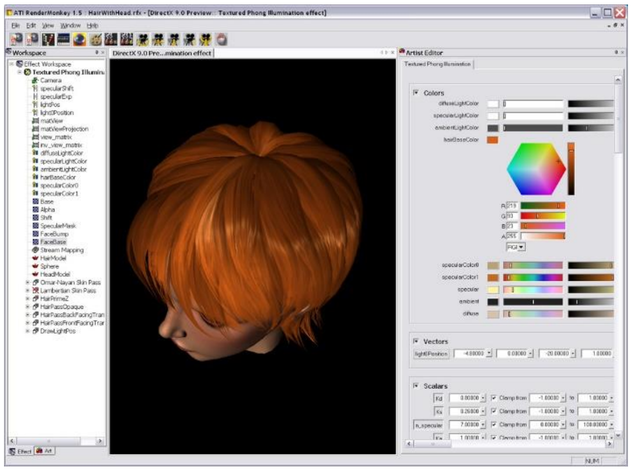
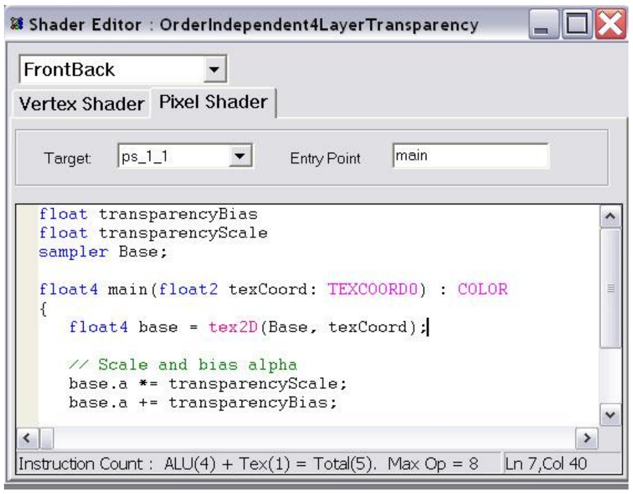


Figure 2.1. Artist in RenderMonkey [Digital image]. (n.d.). Retrieved December 07, 2018, from https://gpuopen.com/archive/gamescgi/rendermonkey-toolsuite/rendermonkey-toolsuite-ide-features/



Target: ps_1_1 Entry Point main

```
float transparencyBias
float transparencyScale
sampler Base;

float4 main(float2 texCoord: TEXCOORD0) : COLOR
{
    float4 base = tex2D(Base, texCoord);

    // Scale and bias alpha
    base.a *= transparencyScale;
    base.a += transparencyBias;
```

Instruction Count : ALU(4) + Tex(1) = Total(5). Max Op = 8 Ln 7,Col 40

Figure 2.2. Shader Editor in RenderMonkey [Digital image]. (n.d.). Retrieved December 07, 2018, from https://gpuopen.com/archive/gamescgi/rendermonkey-toolsuite/rendermonkey-toolsuite-ide-features/

In order to speed up programming, RenderMonkey provided with a list of predefined variables, more commonly known as uniform variables in shader language. The names of these variables could be customized to match those of the specific engine the shaders were being developed in. The default match those of Microsoft FX file format (RenderMonkey Toolsuite, n.d.). However, such a feature will be unnecessary for our tool since it will be developed for the Processing engine only. Moreover, the overall interface of RenderMonkey does not match the simple and minimal interface of Processing Development Environment.

Another important thing we identified after reviewing RenderMonkey was that though it has been developed to support both programmers and artists, it is less helpful to those who are intermediate programmers who will be our main targeted users. Such users will want to make use of both coding and visual elements for prototyping or learn the advanced concepts of shader programming with time. Nevertheless, RenderMonkey is still a great inspiration for our tool in terms of real-time shader programming that does not require repeated compilation of code.

2.5 Live Coding shaders

RenderMonkey is not the only platform that allows real-time shader programming also known as live coding for prototyping purposes. There are many other platforms that allow live coding as well. Live coding refers to the process of code being evaluated and compiled in the background as the user types, leading to quick visual prototyping effects (KodeLife, n.d.). A great advantage of live coding is that it allows continuous and uninterrupted working. This will increase productivity for computational artists as they will have more time to focus on the design of their product and also improve the quality of the final product (Tatarchuk, 2004).

Amongst the platforms that offer live shader coding a few include Shadertoy (Jeremias, 2014), GLSL sandbox (Cabello, n.d.) and Kodelife (KodeLife, n.d.). However, most of these are proprietary software which means that they could not be integrated with a desktop application such as Processing. Though Shadertoy offers a huge database of shaders for artists to get inspiration from, these are often long, complex snippets of code with no documentation, making it hard for an inexperienced user to understand. Moreover, Shadertoy runs slower on most browsers because of having a large database of complex shaders and requires a really good graphics card to run these smoothly (Colubri, n.d.). However, a basic to intermediate shader runs smoothly on Processing’s internal renderer and does not require a high-end PC (Colubri, n.d.). Other drawbacks in Shadertoy and GLSL sandbox include allowing only fragment shaders to be edited. This minimizes the range of effects that can be achieved from manipulation of vertices or geometry (Jeremias, 2014). KodeLife on the other hand allows multiple shaders to be editable but the text and rendered display are in the same window which means that often times the text blends in with the background, making it hard for the user to type (KodeLife). Compared to these, Shadertoy offers a more visually balanced interface with the text editor and rendered view alongside each other (Quilez).

2.6 GLSL Interface for Teaching

While searching for shader programming frameworks that had commonalties with our research goals, we also found some that were built to teach shader programming to users. The audience of most of these were university students who were learning programming. Glman is one such framework (Bailey, 2007). Following the design of the RenderMan Interface Bytestream (RIB), glman reads an input file called GLIB (GL Interface Bytestream) along with some shader files (vertex or/and fragment). These files are then used to generate the desired scene, with sliders created from user defined global parameters (Bailey, 2007). The sliders can be helpful in observing effects of specific parameters instantly hence avoiding the need to recompile the program after every change in value (Bailey, 2007). However, one thing to note is that the glman interface does not support live coding which is one of the main goals of our research. As more and more visual designers and artists are embracing the art of creative coding, having an interface that encourages live coding with minimum distraction such as recompiling shader program is becoming vital. In the glman study, it was hard for us to judge the effect of the program on student's learning curve regarding shaders as their study did not provide details of the experiment or survey that they performed to arrive to the conclusion that "students get a maximum amount of quality learning in the minimum amount of time" (Bailey, 2007, p. 1).

Georgia State University did research to compare three different methods of teaching programmable shaders categorized as heavy-weight, mid-weight and light-weight (Owen, 2005). The heavy-weight approach involved a programming intensive approach in which low-level Cg (computer graphics) program had to be integrated with OpenGL program using the Cg library and traditional OpenGL API (Owen, 2005). Since this focused less on GPU programming and more on the setup of OpenGL, it would not benefit those aiming to learn shader programming. The lightweight approach involved making use of X3D/VRML to teach programming of GPUs (Owen, 2005). X3D, also known as Extensible 3D Graphics, is a standard for representing 3-dimensional models on the web (Brutzman, 2015). All the students had to do was to insert their vertex and fragment shaders inside nodes called ShaderAppearance nodes and load the VRML in a browser. No in-depth knowledge of VRML was needed. However as of now these can only be run in Windows browsers as plugins for non-Windows browsers have not been developed yet. This enforces a platform specific constraint posing a disadvantage to users who work on other platforms. The middle-weight approach on the other hand was a combination of the first two approaches. It included the use of Python language for programming shaders. Since Python does not require compilation at runtime, this meant that users could view the results of changing code without having to stop the initial rendering and recompiling.

An analysis of all three approaches in the context of Processing shows that the middle-weight approach will be least preferred since Python scripts are not supported by the Processing Development Environment. The heavy-weight on the other hand requires in-depth knowledge of OpenGL and Cg which processing users are not expected to know. The lightweight approach can be very useful in the context of processing since it uses OpenGL shading language which is supported by VRML files. However, the task of inserting the shader code from processing sketch shader files into VRML nodes and loading them in a browser to view the results can be tedious and time consuming.

Another similar framework that was built to teach modern computer graphics concepts particularly shaders was glGA (OpenGL Geometric Application) framework (Papagiannakis, 2014). This framework was built with a similar goal to our research, which was to design a simple shader interface for novice users in a way that gradually builds their expertise and understanding of shader programming instead of burdening them with loads of information and features all at once like most books and online tutorials do (Papagiannakis, 2014).

However, this framework was specifically designed to teach a wide range of concepts in the Computer Graphics pipeline including shader writing. However, it did not have support to prototype new shaders. Moreover, the knowledge regarding shader writing was heavily focused on those who wanted to ultimately delve into low-level graphics programming and shader development. Since a great majority of Processing users are computational artists and designers, bombarding them with low-level advanced shader programming concepts can act as an obstacle in their way of creative thinking. That is one of the reasons the Processing API consists of simplified syntax as compared to other graphical frameworks, hiding most of the advanced functionality from the user and maintaining a “balance between features and clarity” (Reas, 2003, p. 1).

The review of graphical interfaces aimed towards teaching concepts of shader programming tells us that though a pedagogical approach in designing/motive behind a shader interface can greatly help computational artists and designers in getting comfortable with shader programming, it alone is not enough to contribute towards making shader programming easier and faster for Processing users. The flexibility to code live in an interface as simple as that of Processing are equally important factors which are missing in glGA but will be a major focus of our research.

2.7 Summary

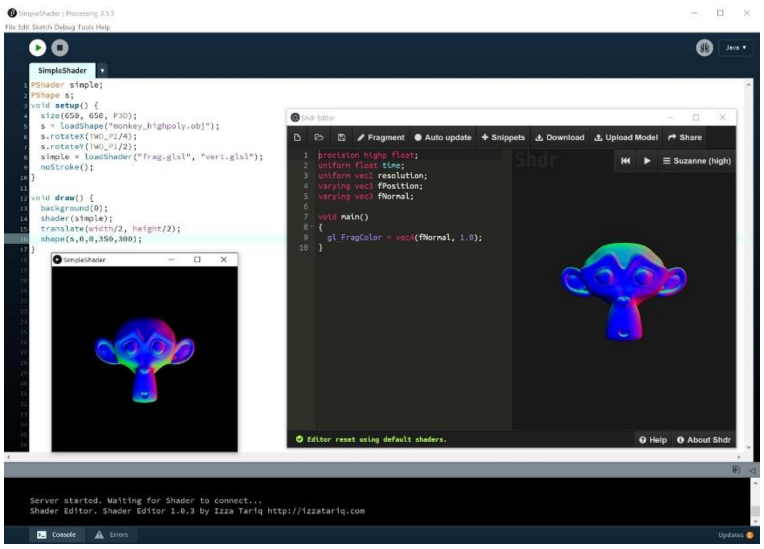
This chapter discussed various shader editors that were relevant to our study. RenderMonkey was a great inspiration for our study. However, it does not cater the needs of intermediate programmers. ShaderToy and KodeLife were also good examples of real-time shader programming but are proprietary softwares with complex shader examples and documentation. Visual shader editors, another option we explored, offered a completely visual approach to creating shaders but will not be favored by computational artists and designers interested in traditional coding. Some of these tools were also designed to teach concepts of shader programming but those concepts were too advanced for computational artists and designers. Though each tool had some favorable features relevant to our study, none of them were completely fit for our targeted users who are computational artists and designers with a basic understanding of programming.

We also found that the Processing Development Environment is a software specifically designed with the goal of teaching fundamentals of computer programming to artists and designers. It also has support for shader programming but lacks a user-friendly interface that can make shader programming easier for its users. Therefore, we have decided to create a new shader tool within the Processing Development Environment.

CHAPTER 3 SHADER MODE IMPLEMENTATION

3.1 Pilot Study

The pilot phase of this study was done as a project for Google Summer of Code (GSOC) in the summer of 2018. Figure 3.1 shows a screenshot of the prototype tool created as part of the project. “The goal of the project was to develop a shader editing tool for the Processing Development Environment (PDE) that updates the PDE sketch display window in real-time, a process known as hot reloading. It was decided to use an existing open-source shader editor as the starting point as building the entire tool from scratch would be impossible to complete within the 3-month time period of GSOC. After exploring several options, an open-source shader editor called Shdr was chosen”. Processing Tool Template served as the starting point for the development of the shader editor tool in Processing (Tariq, 2018).



```
PShader simple;
PShape s;
void setup() {
  size(650, 650, P3D);
  s = loadShape("monkey_highpoly.obj");
  s.rotateX(TWO_PI/4);
  s.rotateY(TWO_PI/2);
  simple = loadShader("frag.glsl", "vert.glsl");
  noStroke();
}

void draw() {
  background(0);
  shader(simple);
  translate(width/2, height/2);
  shape(s,0,0,350,300);
}
```

```
precision highp float;
uniform float time;
uniform vec2 resolution;
varying vec3 fPosition;
varying vec3 fNormal;

void main()
{
  gl_FragColor = vec4(fNormal, 1.0);
}
```

```
Editor reset using default shaders.
```

```
Server started. Waiting for Shader to connect...
Shader Editor. Shader Editor 1.0.3 by Izza Tariq http://izzatariq.com
```

Figure 3.1. Shader Editor in Processing

Since Shdr was a web-based app developed in JavaScript while the PDE is a desktop app developed in Java, direct integration of Shdr was not possible. To resolve this, first a desktop version of Shdr was generated using Electron (a tool to build cross-platform desktop apps) and Node.js. The next step was to figure out how to enable communication between the PDE and Shdr considering that their source code was written in different languages. This was resolved using inter-process communication (IPC) using socket connections. A server-side connection was set up in the Processing tool template using windows sockets while the client-side connection was implemented in Shdr's index.js (the main application file) file using the Node.js NET module.

IPC was initially used to inform PDE to compile its default shader whenever the text editor inside Shdr was updated by the user. However, this was later replaced by a simpler solution in which PDE checked the timestamps of its local shader files to update its display window. The other function of IPC was to send the location of the sketch shader files to Shdr so that Shdr could read from and write to the correct location whenever the user edited the contents in its editor, saved or created a new shader file. Messages were sent over IPC in the form of strings. Once IPC was running smoothly, different case scenarios for using the Shader Editor tool were taken care of. These were:

- creating data folder inside the sketch folder if it was absent
- creating default shader files inside data folder
- storing Shader Editor contents to temp folder if sketch is untitled
- sending updated path of data folder to Shader Editor when the untitled sketch is saved

Running a simple shader generated two rendering displays as shown in Figure 3.1, one belonging to the PDE (left) and the other belonging to the Shdr editor (right).

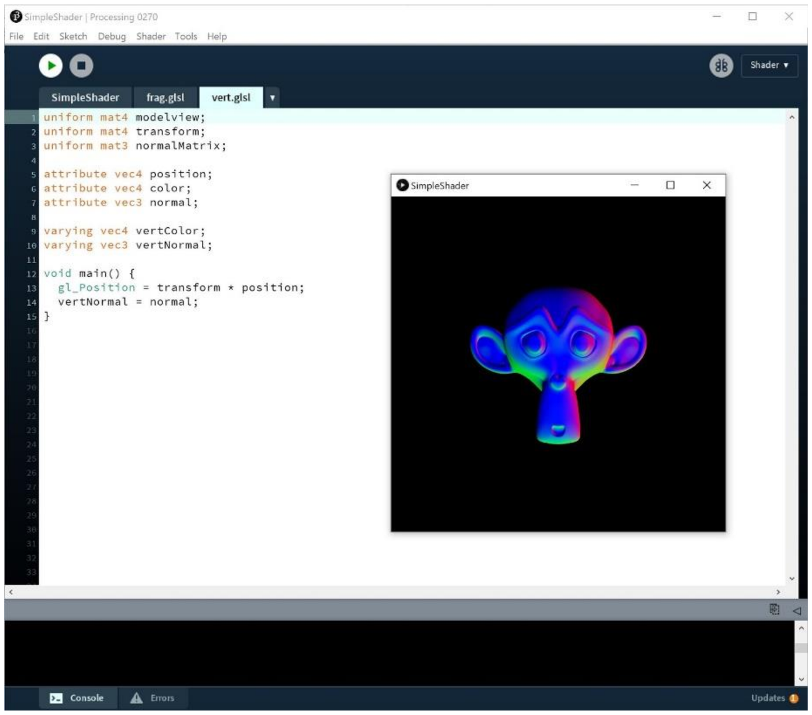
However, this implementation of shader editor did not work fine every time and generated errors or gave inconsistent results in a number of cases. Firstly, it did not provide a smooth workflow as the shader editor had to be run in a specific order which if not followed generated errors. Secondly, it led to inconsistent rendering displays because of conflicting naming conventions of default shader uniform variables and attributes in the Shdr and Processing API. Thirdly, custom uniform variables could not be passed from the main sketch file to Shdr in real-time as the Processing Tool Template did not have access to contents of the Processing sketch files. Furthermore, Shdr did not have access to geometry created in the main PDE sketch file. Rather it was programmed to use its default stored meshes to show the effect of shader code. Due to this Shdr's rendering display was often inconsistent with that of Processing's rendering display.

3.2 Current Study

The processing tool template initially used to develop the shader editor in processing gave very limited access to Processing API's internal classes as was mentioned previously. This made it almost impossible to modify the PDE's existing interface to support editing of shader files. This was the main reason for integrating an external shader editor that came equipped with a text editor, shader compiler and renderer. However, the integration did not turn out to be completely successful as discussed at the end of the previous section.

Though some of the issues could be solved in the previous implementation of shader editor in Processing, it was realized that there was a much better and simpler approach that did not involve integrating an external editor. This new approach was based on the idea of modes in Processing. Processing has different programming modes, such as Java, Python, Android and p5.js (JavaScript) that allow flexibility with different types of platforms and programming techniques. Thus, it made sense to have one for shader programming as well. Since a mode in PDE can be customized to open different types of file formats, users could create and edit shader files within the same workspace. There would be no foreign editor involved, simplifying the development process. Moreover, the multi-tab feature in the PDE would also facilitate switching between shader and non-shader files (PDE sketch files) hence leading to smoother navigation.

Shader Mode is a contributed mode that inherits directly from the current Java Mode in Processing. Hence it comes with most of the features already present in the Java Mode. Additional features to aid in shader programming were then added to this new mode. These included (a) create new shader file (b) syntax highlighting of glsl code, (c) Moving shader files to root sketch folder (d) a shader specific menu (e) shader templates (f) shader examples (g) pop-up reminder for post-survey. The function and purpose of each feature is further explained in the sections below. Figure 3.2 shows a final working version of the Shader Mode.


SimpleShader | Processing 0270
File Edit Sketch Debug Shader Tools Help
Shader ▾
SimpleShader
frag.glsl
vert.glsl
SimpleShader
Console
Errors
Updates

```
uniform mat4 modelview;
uniform mat4 transform;
uniform mat3 normalMatrix;

attribute vec4 position;
attribute vec4 color;
attribute vec3 normal;

varying vec4 vertColor;
varying vec3 vertNormal;

void main() {
  gl_Position = transform * position;
  vertNormal = normal;
}
```

Figure 3.2. Shader program running in the Shader Mode tool in PDE with multiple shader files (frag.glsl and vert.glsl) opened in separate tabs. The smaller sub-window shows a rendered model of the Blender monkey head "Suzanne".

3.2.1 Create New Shader File

A new shader file can be created by using the "Create New Tab" option or the new file option in the File menu. The .glsl extension has to be provided with the name as shown in Figure 3.3, otherwise the type of the file defaults to .pde. This feature was implemented by allowing files with the .glsl extension to open up in the mode by overriding a function called "getExtensions()" so that it returned an array including "glsl".

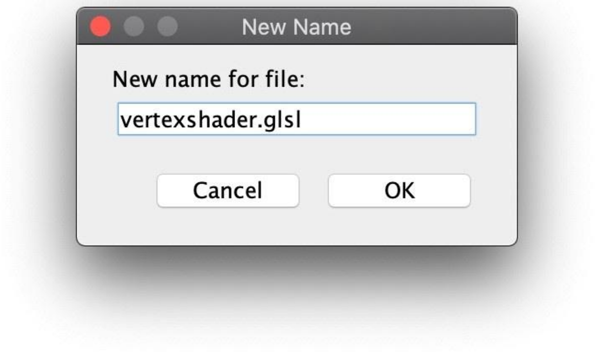
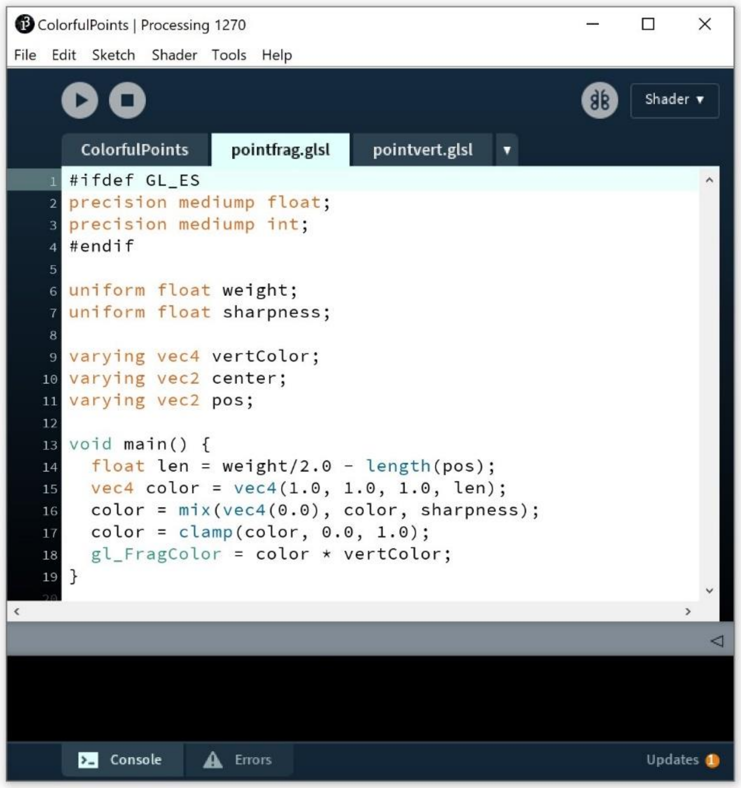


Figure 3.3. Dialogue box for entering name for new shader file

3.2.2 Syntax Highlighting

Syntax highlighting is included in text editors to make the process of coding easier. Coloring of text makes it easier to notice syntactic (Sarkar, 2015). Hence the purpose of implementing syntax highlighting of GLSL code in Shader Mode was to allow better code readability and prevent syntax errors. Syntax highlighting was implemented by creating a text file for GLSL keywords that were divided into different categories such as datatypes, functions, structures, built-in variables etc .The Processing Token Class was used to apply different colors on keywords depending on which category they belonged. An example of a syntax-highlighted shader file is shown in Figure 3.4.



```
#ifdef GL_ES
precision mediump float;
precision mediump int;
#endif

uniform float weight;
uniform float sharpness;

varying vec4 vertColor;
varying vec2 center;
varying vec2 pos;

void main() {
  float len = weight/2.0 - length(pos);
  vec4 color = vec4(1.0, 1.0, 1.0, len);
  color = mix(vec4(0.0), color, sharpness);
  color = clamp(color, 0.0, 1.0);
  gl_FragColor = color * vertColor;
}
```

Figure 3.4. A Processing shader file showing syntax-highlighting of GLSL code

3.2.3 Moving shader files to root

Previously Processing shader files had to be placed in a separate “data” folder inside the project root folder. However, this restriction was removed later on and users are now encouraged to keep the shader files in the project root directory. The current Shader Mode tool also saves new shader files in the root directory. However, when opening existing shader files, it relocates the shader files to the project root directory if they were initially present in the data folder. The user is asked for permission before carrying out this operation so that they are aware of the change.

3.2.4 Shader Menu

A shader menu was created alongside existing menu items. Currently this only contains a few items such as link to online tutorial, shader templates, link to post-survey. More options can be added in a future release. Figure 3.5 shows a screenshot of Shader Menu items.

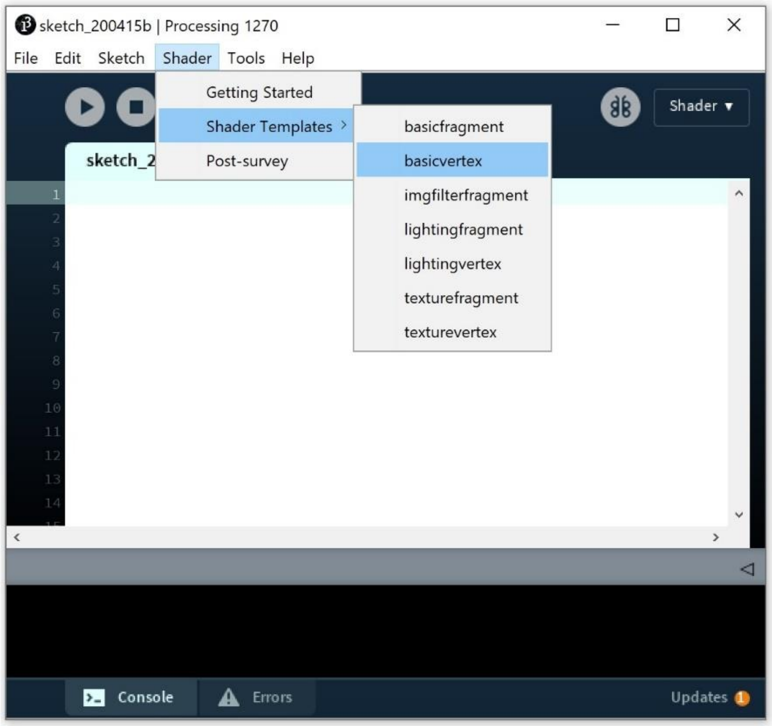
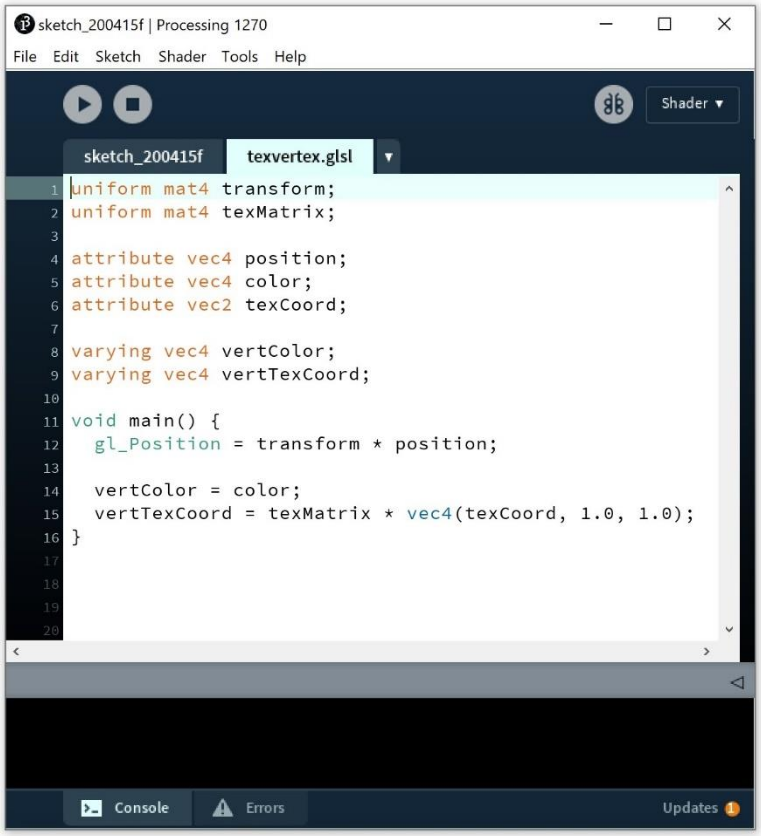


Figure 3.5. Screenshot of Shader Menu in Shader Mode

3.2.5 Shader Templates

Shader templates are GLSL files containing some shader code to give users a starting point to develop different types of shaders. This can sometimes save time setting up shaders especially if users forget the exact names of default shader uniform variables or attributes specific to the Processing API and have to search for them online. The current version of Shader

Mode contains seven templates. The exact content of the “texvertex” template is shown in Figure 3.6. This template is meant to be used for texturing a scene or object.


sketch_200415f | Processing 1270
File Edit Sketch Shader Tools Help
Shader
sketch_200415f
texvertex.glsl
Console
Errors
Updates

```
uniform mat4 transform;
uniform mat4 texMatrix;

attribute vec4 position;
attribute vec4 color;
attribute vec2 texCoord;

varying vec4 vertColor;
varying vec4 vertTexCoord;

void main() {
  gl_Position = transform * position;

  vertColor = color;
  vertTexCoord = texMatrix * vec4(texCoord, 1.0, 1.0);
}
```

Figure 3.6. Template for setting up a vertex shader for texturing in Shader Mode

3.2.6 Shader Mode Examples

Shader examples can be a great resource for both experienced and less experienced users. Examples serve as demos for inspiration for new ideas for users. Every Processing Mode comes equipped with its own examples, thus it made sense to include some for the Shader Mode as well. These examples can be accessed from the File menu. Since our targeted community includes both experienced and less experienced users, we added a set of both basic and advanced shader examples as is shown in Figure 3.7.

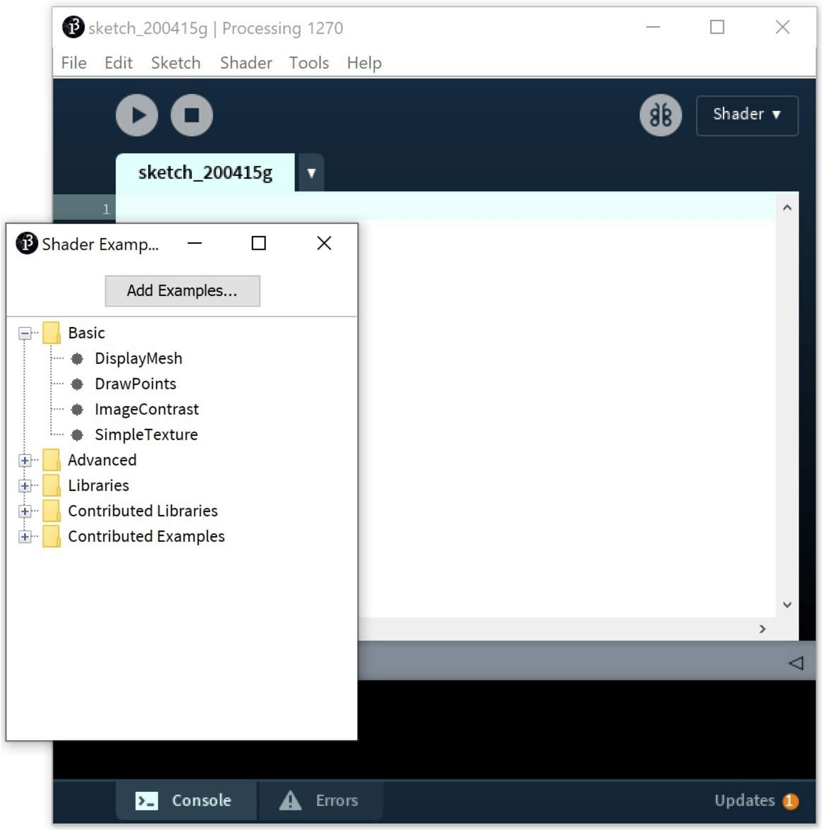


Figure 3.7. A sub-window showing Shader Mode Basic and Advanced Examples

3.2.7 Pop-up reminder

In order to ensure that users did not forget to fill and submit the post-survey, a pop-reminder was added that showed up every time users attempted to close the PDE window.

CHAPTER 4 METHODOLOGY

4.1 Sample Set

Considering that our tool was specifically developed for the Processing Development Environment, we decided to restrict our users to the Processing community. Since Processing has a much larger community online than in one specific place, the online community was considered to be more accessible. Hence, we reached out to volunteers through different social platforms such as the Processing Forum, Processing's Instagram page and Processing's twitter handle. An online invitation for participation in our study was posted on these social platforms. Upon accepting the invitation, users were asked to fill out an online pre-survey.

Apart from being part of the Processing community, users were also required to have some experience with shader programming. One thing to note is that the number of users familiar with programming shaders in Processing is quite small as compared to the overall Processing community. This was one of the major reasons that made it difficult to get enough participants for our study. Even though 30 users signed up for our study, only 8 users filled and submitted the post-survey.

4.2 Experimental Setup

The invitation for the study was posted on different social media platforms with a sign-up link for the study in description. Consent for participation was included in the online invitation form. After signing up, users were directed to a pre-survey. The survey asked users to self-report their expertise in shader programming from a scale of 1 to 5 as well as provide their email address. The email addresses provided were saved in our database.

Further instructions regarding the experiment were emailed to each user. The exact content of the email can be found in the appendix. The email also contained a link to the downloadable version of Shader Mode with installation instructions and a link to the latest version of processing (Processing 4.0 alpha) which Shader Mode supports.

Users were required to perform the experiment on their personal PCs and spend between 30-45 minutes. The task for the experiment was open ended meaning participants were not given a fixed set of instructions to accomplish or create something. Instead they were asked to navigate the Shader Mode interface as they liked by running different scenarios they might have in mind. Examples of different scenarios mentioned included writing a new custom shader in the PDE, opening and editing an existing shader in the PDE etc. Participants were also informed about the default shader examples in the PDE and were encouraged to take inspiration from them if needed. Upon completion of the task users were required to fill a post-survey, the link to which was provided in the email as well as the Shader Mode menu.

4.3 Data Collection

Once the required number of survey responses were collected, data was anonymized i.e. the name or identity of the users was removed from the data. The data collected from the surveys was both qualitative and quantitative. The answers to rating-based questions generated numeric responses that were automatically stored in an online excel spreadsheet. This was our quantitative data. The qualitative data comprised text-based responses to two short-answer questions asking them to reflect on their experience and provide any suggestions or improvements.

Google Forms was used to create the online survey as it is an affordable and easy to use tool for both researchers and participants (Binyamin, 2016). Responses from the form were automatically stored online in a linked google sheet. All data collected was kept safe in a password protected computer with scheduled backups.

4.3.1 System Usability Scale

The goal of our study was to determine the usability of a real-time shader editor inside Processing for emerging computational artists. Since we were interested in measuring perceived usability, we chose System Usability Scale (SUS) survey as our usability questionnaire. SUS is a 10-item questionnaire designed to measure user's perceived usability (Garcia, 2013). It provides usability in the form of SUS scores.

Many studies have found SUS to be an accurate and reliable tool even for small sample sizes (Binyamin, 2016). Another advantage of using SUS is that it is also a reliable measure of perceived learnability (SUS: The System Usability Scale, n.d.). This is very important in regard to our study since one of the goals of our tool is to help promote learning of shaders for novice computational artists. Moreover, SUS is more precise as compared to other surveys such as Software Usability Measurement Inventory (SUMI) which has 50 questions and Post-Study Usability Questionnaire (PSSUQ) which has 16 questions. Since our study was to be conducted remotely, we had less control over making sure that participants completed the experiment as we could have had in an in-person study. Therefore, we wanted to avoid demanding too much of their personal time as it could make them lose interest in the study.

Although SUS provides a measure of overall usability of a tool, specific usability issues cannot be identified by SUS as SUS is not a diagnostic tool (Binyamin, 2016). Therefore, we added short answer based open-ended questions at the end of the survey to get user's feedback about specific features of the tool. This would also help us get a better understanding of the quantitative data. The complete form we used for our study can be found in the Appendix section.

4.4 Data Analysis

After data collection, data was downloaded from the online google sheet to a Microsoft Excel file on the desktop for statistical analysis.

Although the initial plan was to perform hypothesis testing for our study, we could not continue with that later in the study as our sample size came out to be too small. The minimum sample size required for hypothesis testing to generate statistically sound results for our study with an effect size of 0.5 and confidence level of 95% was approximately 30. However, we only managed to get 7 users. Hence, we only used descriptive statistics to analyze the quantitative data for the given sample.

The statistical analysis began with calculating a SUS score for each user. From the resultant SUS scores, statistics such as mean and standard deviation were calculated. Graphs and charts were drawn to further illustrate the data. The statistical data was further elaborated upon by doing a qualitative analysis of the responses to short-answer questions. These results were only valid for the given sample and were not generalizable to the larger population.

4.5 Validity

Using the SUS survey ensures construct validity in our study. “SUS has been shown to effectively distinguish between unusable and usable systems as well as or better than proprietary questionnaires. SUS also correlates highly with other questionnaire-based measurements of usability” (Sauro, 2016). In order to further validate the results obtained from SUS survey, we will compare the quantitative survey results with the text-based responses given by users. This comparison will help us establish a relationship between both the qualitative and quantitative results and enable us to get a better understanding of the quantitative data.

CHAPTER 5 PRESENTATION OF DATA & FINDINGS

5.1 Quantitative Analysis

As mentioned before, SUS provides usability in the form of SUS scores. SUS scores for each user are calculated using the following steps:

1. Subtract 1 from the score of each odd numbered question.
2. Subtract the score of each even numbered question from 5.
3. Add scores of all questions and multiply by 2.5 to get scores in the range 0-100.

The last step does not generate percentages and are considered only in terms of their percentile ranking.

Figure 5.1 shows the frequency distribution of SUS scores from our study. All scores were unique except for a SUS score of 67.5 that was generated for two users.

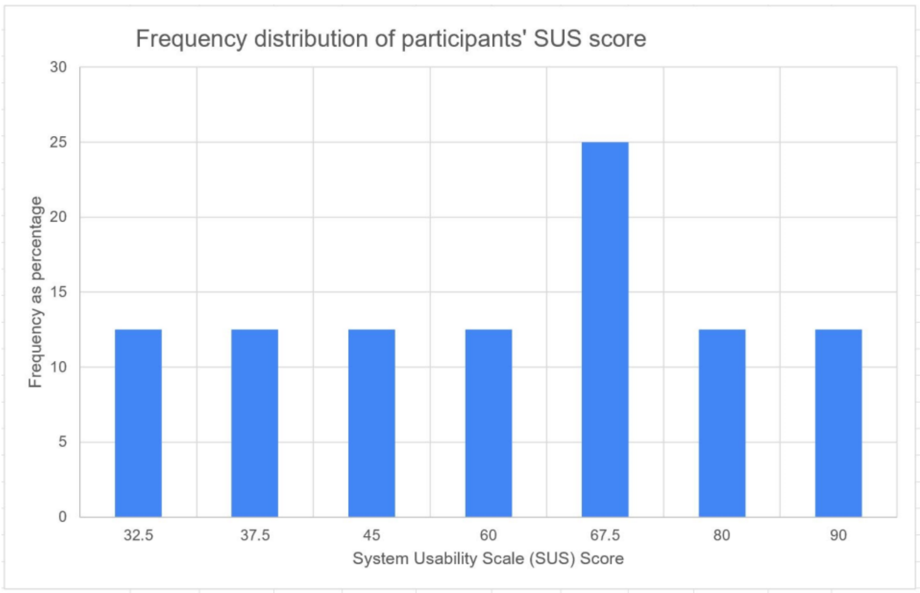


Figure 5.1. Mean normalized SUS scores of 8 participants

Once the SUS scores were finalized, mean and standard deviation of the scores were calculated for 3 different conditions as shown in Table 5.1. Expertise of each user was self-reported from a scale of 1 to 5. Shader expertise rated below 3 was considered low while that rated equal to 3 or above was considered high. As seen in Table 5.1, 3 of our users reported low expertise in shader programming while the remaining 5 reported high expertise. The mean SUS score for users with low expertise came out to be lower than users with high expertise in shader programming. Both groups when combined gave a mean SUS score of 60. This is further illustrated in a bar chart in Figure 5.2.

Table 5.1. Descriptive statistics for Shader Mode usability

Condition	Sample size (N)	Mean	Std. Dev
Low Expertise	3	43.33	14.65
High Expertise	5	70	16.86
Combined	8	60	20.35

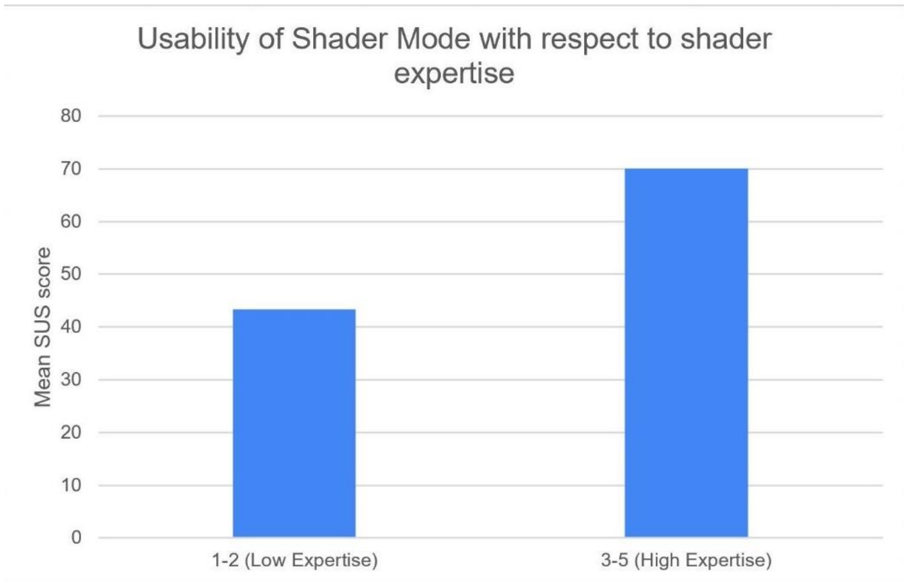


Figure 5.2. Bar chart showing usability of Shader Mode with respect to shader expertise

In order to evaluate the usability of the SUS scores in our study we used a scale for acceptable SUS scores that provided statistically valid results in a study focused on learning management systems (Binyamin, 2016). The scale the said study used was originally provided in another study (Bangor, 2008). We present the scale in the form of a table as shown in Table 5.2.

Table 5.2. Scale for acceptable SUS scores

SUS score	**Usability**
85 - 100	Highly usable
70 - 85	Excellent
50 - 70	Good (users experienced usability issues)
0 - 50	Not acceptable

5.2 Qualitative Analysis

As mentioned previously, our survey also collected text-based responses to short-answer questions to help us get a better understanding of the quantitative results and to get possible insight into particular aspects of the tool that contributed to those results. It should be noted that in general "survey written open responses are generally not expected to provide the depth of information that interview or focus group data would provide". Hence this qualitative analysis is just an illustrative addition to our statistical analysis and is meant to explain the reasoning behind the quantitative results.

We used open coding to analyze the survey responses. Open coding is a way of inducing themes from text (Ryan, 2000). We primarily used word frequencies and clustering techniques. Table 5.3 shows a table with different themes identified in responses from each of the eight different users. These themes were the result of a series of consecutive steps: eye-balling the text, finding, and listing important words and verbs, grouping into categories and sub-categories. After that we also generated the frequencies of the most popular terms relevant to our study as shown in Table 5.4. The results were sorted from high to low for clarity and understanding.

Table 5.3. Results of manual open coding on survey comments

SUS score	expertise	liked/useful	disliked/ drawbacks	Usage	preference over other editors	Lacking / improvements	feeling
32.5	2	syntax highlighting	bugs -> poor debugging, not clear			better error check	frustration, confusion
37.5	1	link to online tutorial		learning of shaders	equal	more examples with baby steps	
45	3	examples			prefer KodeLife	live coding	
						abstraction of interface	
60	2	-	-	-	-	-	-
67.5	3					tutorials, documentation	
67.5	4	syntax highlighting	none			tutorials	enjoyed
		creation of new GLSL files				more examples	
80	3	syntax highlighting		teach students		more comments in examples for beginners	
		templates				include more basic variables in templates	
90	4	syntax highlighting, templates, examples				debugger, documentation	

Table 5.4. Frequency of popular terms used

Terms	frequency
examples	4
syntax highlighting	3
tutorials	3
templates	1

It was observed that the features that caught the most attention from users was the syntax highlighting of GLSL language and default shader examples. While everybody approved of syntax highlighting, comments regarding Shader Mode examples fell on two sides: those that regarded shaders to be incredible and very useful and those that pointed to the need for more examples with better comments and explanations. Of the latter, even users that reported high expertise in shader programming also suggested it as a need for beginners or those new to shader programming.

Two users also pointed at the need for an integrated tutorial for shaders so that they did not have to refer to go for help online. We only got two responses highlighting the learning aspect of the tool. Moreover, since both comments conflicted with each other it is hard to predict whether the tool helped in providing learning of shaders.

5.3 Interview

User testing was concluded with an additional phone interview with a shader expert. The interview lasted 45 minutes and took the form of a casual conversation based on open-ended questions that revolved around the interviewee's experience using the Shader Mode. Since the interviewee was not a regular Processing user, their view was not considered strong enough to be generalized to users who were more frequent users of Processing. Nonetheless, it did provide an insight into possible strengths and weaknesses of the tool and clues to future research.

The interviewee stated that they overall had a positive experience using the Shader Mode. They liked that the mode was based on the existing architecture of processing since there are other modes in other processing with different purposes such as the Java Mode, Android Mode etc. This made it easier for them to set up the new mode and have some intuition about its working. They also commented on the integration of the tool, saying that changing the mode did not remove the code from the sketchpad which was good. Lastly, they also liked that Shader Mode came with its own examples that were only visible in that mode.

Due to less familiarity with Processing, the interviewee had to spend two hours on the online tutorial on Processing shaders before spending an additional hour experimenting with the Shader Mode itself. When asked if they would have liked a better tutorial, they responded that they were satisfied with the original explaining that it would be helpful for any new user wanting to get familiar with shader programming in Processing.

One main suggestion that the interviewee had for a future version of Shader Mode included the following:

- Detailed documentation for Processing shader API
- Shader debugger

Though our interviewee called themself to be an expert in shader programming, they did not consider themself to be an expert in Processing. The last time they used processing was seven years ago. Moreover, in their experience with using Processing, they never used it to program shaders. Furthermore, the interviewee also commented that they were not sure if they would prefer Shader Mode over a basic text editor such as Sublime (A sophisticated text editor for code, markup and prose, n.d.) to edit shaders. Rather they would use other more advanced editors to code shaders since in their opinion Shader Mode or even Processing was not designed to support advanced shader programming.

CHAPTER 6 DISCUSSIONS AND CONCLUSION

6.1 Discussion

In this section we will combine the results from the quantitative and qualitative analysis, find connections between them and interpret the results. Moreover, we will use those results to answer the two research questions of our study mentioned in chapter 1.

1. Does Shader Mode have a high usability for all Processing shaders familiar with shader programming?

Our study comprised a total of 8 users where 3 users reported low expertise in shader programming while 5 users reported high expertise in shader programming. Looking back at Table 5.1 when we compare the mean SUS scores with the acceptable SUS score scale, we found that the overall usability of Shader Mode was considered good but users experienced some issues. This was supported by qualitative data where we observed that almost all users either reported facing some issues or felt the need for better features such as more examples, better tutorials and templates. Though only one user expressed some frustration with using the tool, every other user appreciated some aspect of the tool as well as providing suggestions for improvements. Though suggestions do not necessarily mean that users were dissatisfied with their experience of using the tool, it does show that they could improve the experience and hence usability of the tool. As seen from Figure 5.1, 37.5% of our users considered Shader Mode to be above average or good while an additional 25% considered it to be excellent. This leaves a small portion of users who considered it to be below average.

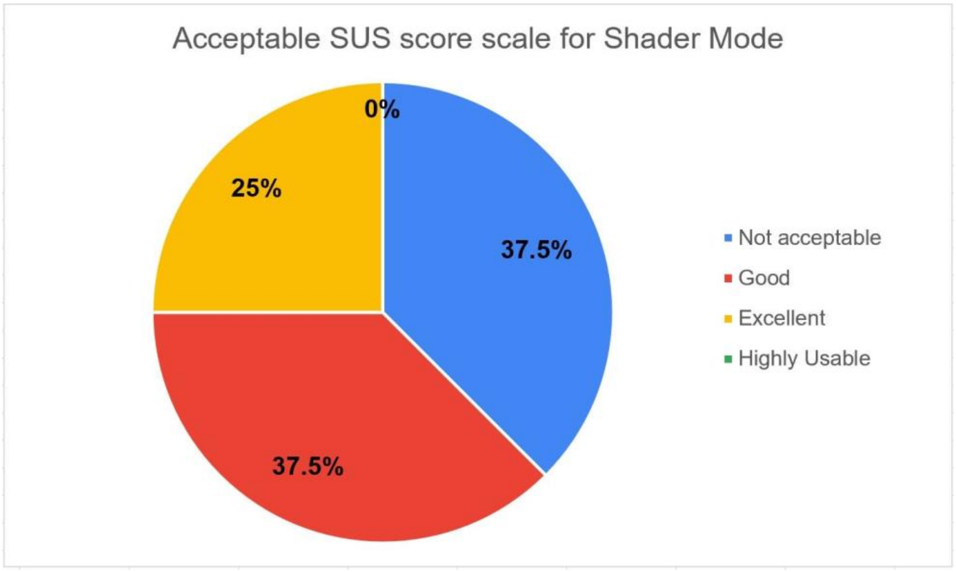


Figure 6.1. Distribution of acceptable SUS score for Shader Mode

Hence to answer our first research question, we can say that the overall usability of Shader Mode was not high. However, at the same time users felt the tool was promising.

2. Is the usability of Shader Mode dependent on the expertise of users in shader programming?

The usability of Shader Mode was found to be quite low for users with low expertise as the mean SUS score for these users was 43.3. According to the scale of acceptable SUS scores, this is considered unacceptable. The qualitative data showed that one user who was fairly new to shader programming did not understand the Shader Mode examples and commented that a step by step explanation of each instruction would be helpful. It seemed that these users did not find the online tutorial much helpful or had difficulty following it. Another user reported facing difficulty setting up shaders in the main sketch program. In general, these users were confused as to what was going on in the mode.

On the other hand, the usability of Shader Mode for users with moderate to high expertise in shader programming came out to be between good to excellent (mean SUS score of 70). Almost all of these users appreciated the syntax highlighting feature and regarded it very useful. They also liked the examples specific to Shader Mode, while at the same time highlighting the need for more basic examples with explanations for less experienced users. These users also mentioned the usefulness of templates, a feature that less experienced users most probably failed to notice or did not know how to make use of.

From the given data and results it seems that user's expertise influenced their perceived usability of Shader Mode. Hence the usability of Shader Mode was dependent on shader expertise of our sample users. It makes sense that users with low shader expertise would find the tool less usable as they were not that familiar with shaders and required more help than was provided to write a complete basic shader. On the contrary users with high expertise in shader programming found the tool to be quite helpful and found it easy to make use of all its features including templates. This also shows that templates are more useful for experienced users as they are already aware of default shader variables and their purpose.

Additionally, it was also noted that three of the users mentioned that they were not sure if the current tool was much different than a basic text editor with a GLSL plugin. This was mentioned by users who had had experience using other GLSL editors. One of them also mentioned that they found KodeLife (KodeLife, n.d.) better than Shader Mode as it featured live coding and hid the interface from the user. Our interviewee also mentioned that they would prefer using more advanced editors for advanced shader programming.

In general, most of the suggestions were regarding including more examples and integrated tutorials to help new users get a better understanding of shaders. The data also suggests that shader examples played an important part in the usability of Shader Mode both in terms of providing inspiration for experienced users and providing learning for less experienced users. Improving this feature could increase the usability of Shader Mode in the future. Moreover, adding comments in templates could be helpful for less experienced users who do not have a good understanding of shader programming concepts. Lastly, having an online documentation for Processing Shader API would be helpful for all users as suggested by our interviewee Jorge. This way users would not have to spend too much time searching for a particular function or variable when they can find it all in one organized place.

6.2 Limitations for the Study

Contrary to our proposed target sample size of 50-100 users, we only received responses from eight users. We believe this was due to the current disruption in world activity that resulted from the outbreak of COVID-19.

Because of a small size we could only use descriptive statistics to analyze our results. Since we cannot infer results using descriptive statistics, we were not able to generate conclusions for the larger Processing community familiar with shader programming.

It was also noticed that there was a difference in the number of users with low and high expertise. Although this difference does not have much of an effect for a larger sample size it can generate inaccurate results for a sample size as small as 8 users. Only 3 users with low expertise participated in our study. A larger number could have affected the usability score for users with low expertise in shader programming.

Due to a very small size, we were also unable to form a control group for the study. The addition of a control group that performed the experiment using the traditional way in Processing (using Java Mode) would have generated comparable SUS scores. This would have helped in finding whether the Shader Mode was more usable than the old method or not.

A qualitative study is usually effective with in-depth interviews or focus groups. Our study included only one such interview. Since Processing users come from a wide range of backgrounds, one interview was not enough to tell how usable our tool was. Our interviewee was a software engineer and an advanced shader programmer but was not a regular Processing user nor had ever used Processing for teaching or learning purposes. Hence, we could not get in-depth feedback from experts in the other domains.

6.3 Future Work

Emerging computational artists should not be required to go through the same rigorous technical process of setting up and coding shaders as advanced low-level programmers do. The idea of templates partially addressed this problem However, as was seen from our study some users wanted help setting up shaders in the main sketch file as well. Hence in the future adding a feature for creating a shader sketch project could be considered. The idea is that this project would come equipped with relevant files, default instructions and variables required to set up any basic shader in Processing. It would allow users to focus more on the creative aspect of their work than the technical aspect of it. As suggested by our interviewee, a custom shader debugger for Processing could also be helpful for figuring out where and what errors if any existed in shader code.

Shader programming greatly expands the possibilities in real-time graphics programming that computational artists can make use of. It is important that such users have access to tools that help them overcome the large technical barrier they face with regards to programming shaders and encourage them to focus more on their creative ability. The new shader mode tool presented in this paper was developed with the goal of bridging the gap between the creative ability of computational artists and the technical expertise expected of them. Although results of the usability are not conclusive, this research will continue as a possible future implementation for the Processing 4 project.

APPENDIX A. ONLINE INVITATION

Processing Forum / Instagram

Hi Processing Community! We are currently conducting a research study on the usability shader programming in the PDE (Processing Development Environment). The study would be conducted remotely and will involve filling a pre-survey, implementing a small coding task in the PDE using your personal computer and sharing your experience using the new mode by filling a post-survey. If you are interested in participating in the study, please sign up using the link below.

https://forms.gle/cfcFESknw45mAXET9

If you have any questions, please feel free to contact me Izza Tariq at itariq@purdue.edu, or the principal investigator of this study, Esteban Garcia Bravo at garcia0@purdue.edu. We appreciate your assistance!

Twitter

Hi Processing Community! We are currently conducting a research study on the usability of shader programming in the PDE (Processing Development Environment). This is an invitation to participate in the study. If you are interested, please sign up using this link:

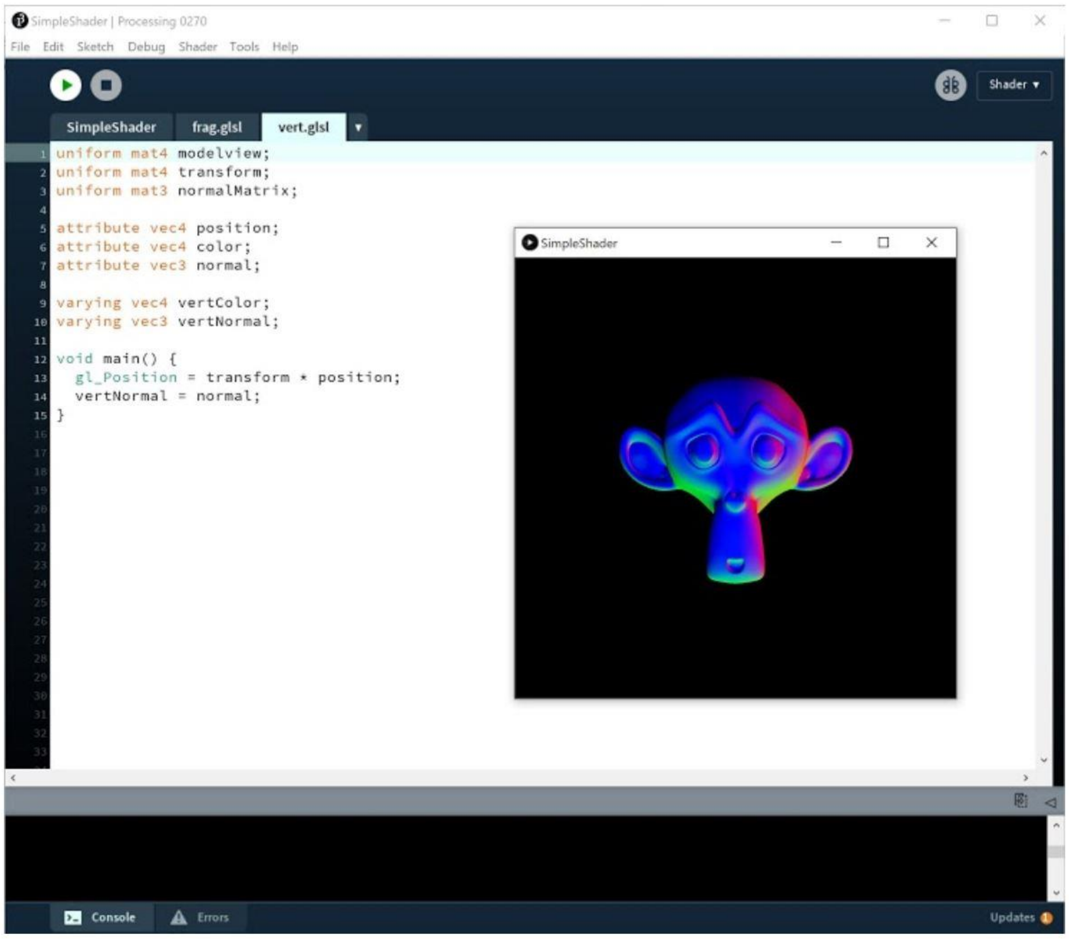
https://forms.gle/cfcFESknw45mAXET9

APPENDIX B. SURVEYS

<u>Pre-survey</u>

Interactive interface for shader programming

* Required

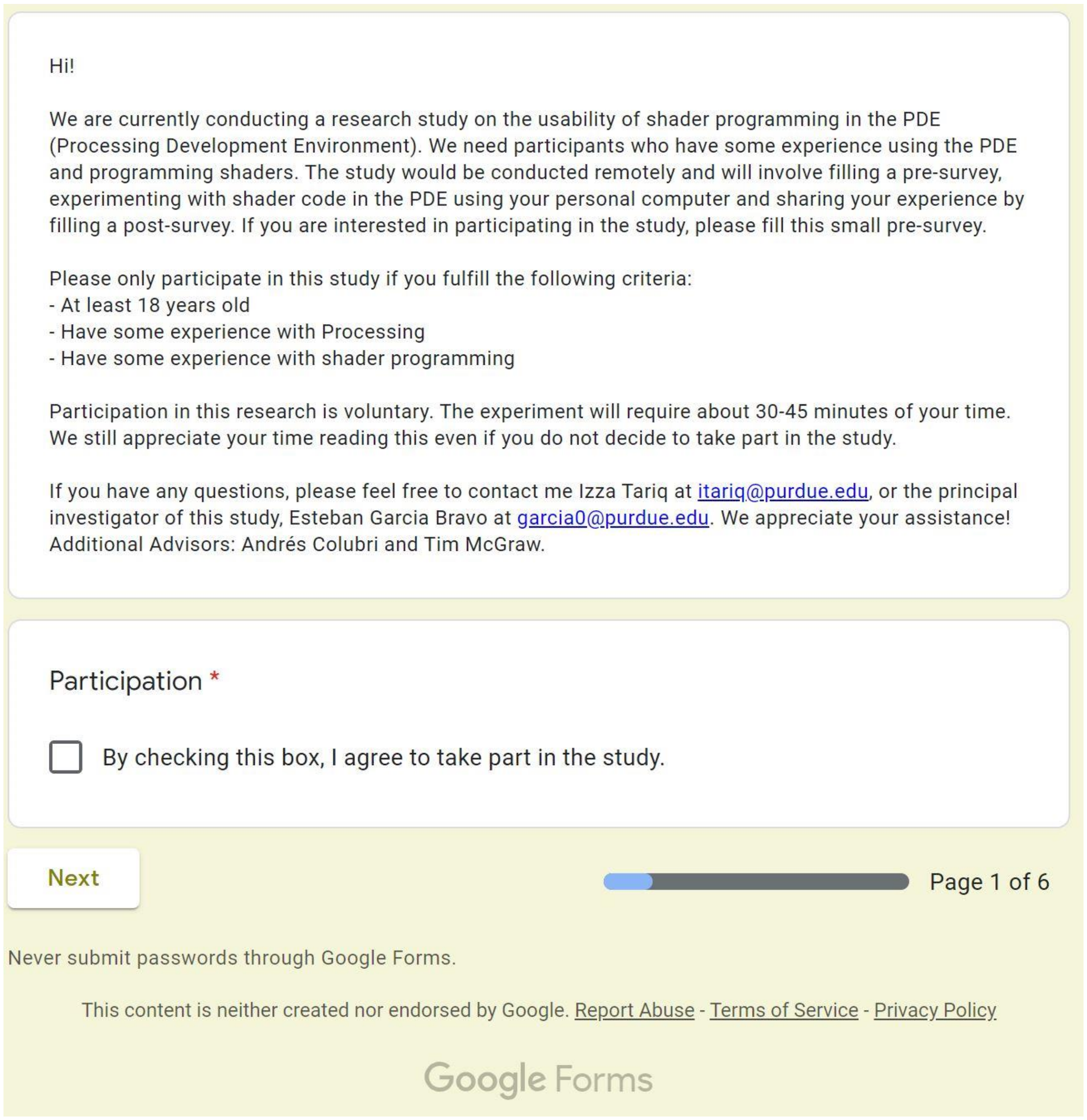

SimpleShader | Processing 0270
File Edit Sketch Debug Shader Tools Help
Shader
SimpleShader frag.glsl vert.glsl
SimpleShader
Console Errors
Updates

```
uniform mat4 modelview;
uniform mat4 transform;
uniform mat3 normalMatrix;

attribute vec4 position;
attribute vec4 color;
attribute vec3 normal;

varying vec4 vertColor;
varying vec3 vertNormal;

void main() {
  gl_Position = transform * position;
  vertNormal = normal;
}
```



Hi!

We are currently conducting a research study on the usability of shader programming in the PDE (Processing Development Environment). We need participants who have some experience using the PDE and programming shaders. The study would be conducted remotely and will involve filling a pre-survey, experimenting with shader code in the PDE using your personal computer and sharing your experience by filling a post-survey. If you are interested in participating in the study, please fill this small pre-survey.

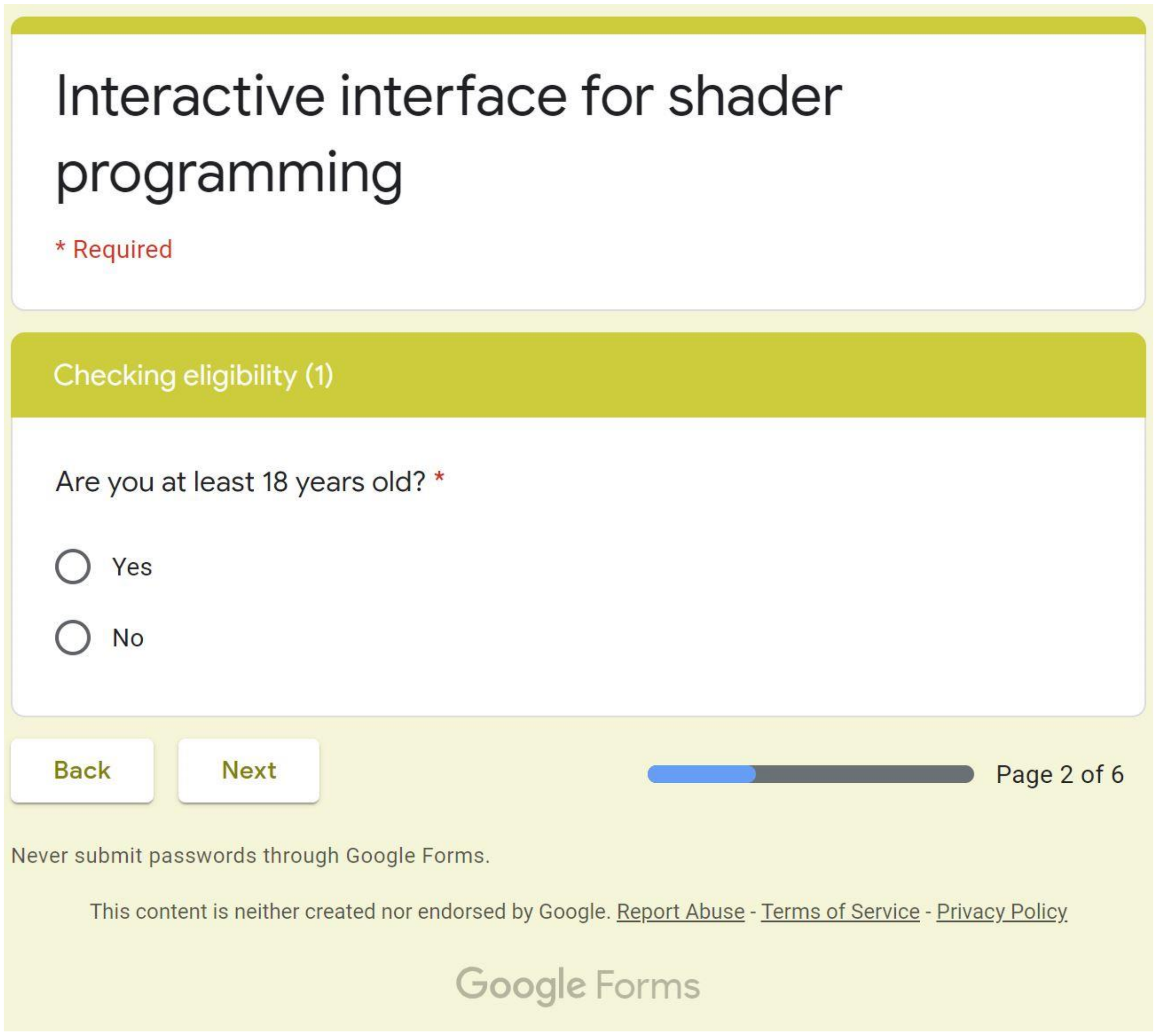
Please only participate in this study if you fulfill the following criteria:
- At least 18 years old
- Have some experience with Processing
- Have some experience with shader programming

Participation in this research is voluntary. The experiment will require about 30-45 minutes of your time. We still appreciate your time reading this even if you do not decide to take part in the study.

If you have any questions, please feel free to contact me Izza Tariq at itariq@purdue.edu, or the principal investigator of this study, Esteban Garcia Bravo at garcia0@purdue.edu. We appreciate your assistance! Additional Advisors: Andrés Colubri and Tim McGraw.

Participation *

☐ By checking this box, I agree to take part in the study.



Interactive interface for shader programming

* Required

Checking eligibility (1)

Are you at least 18 years old? *

○ Yes

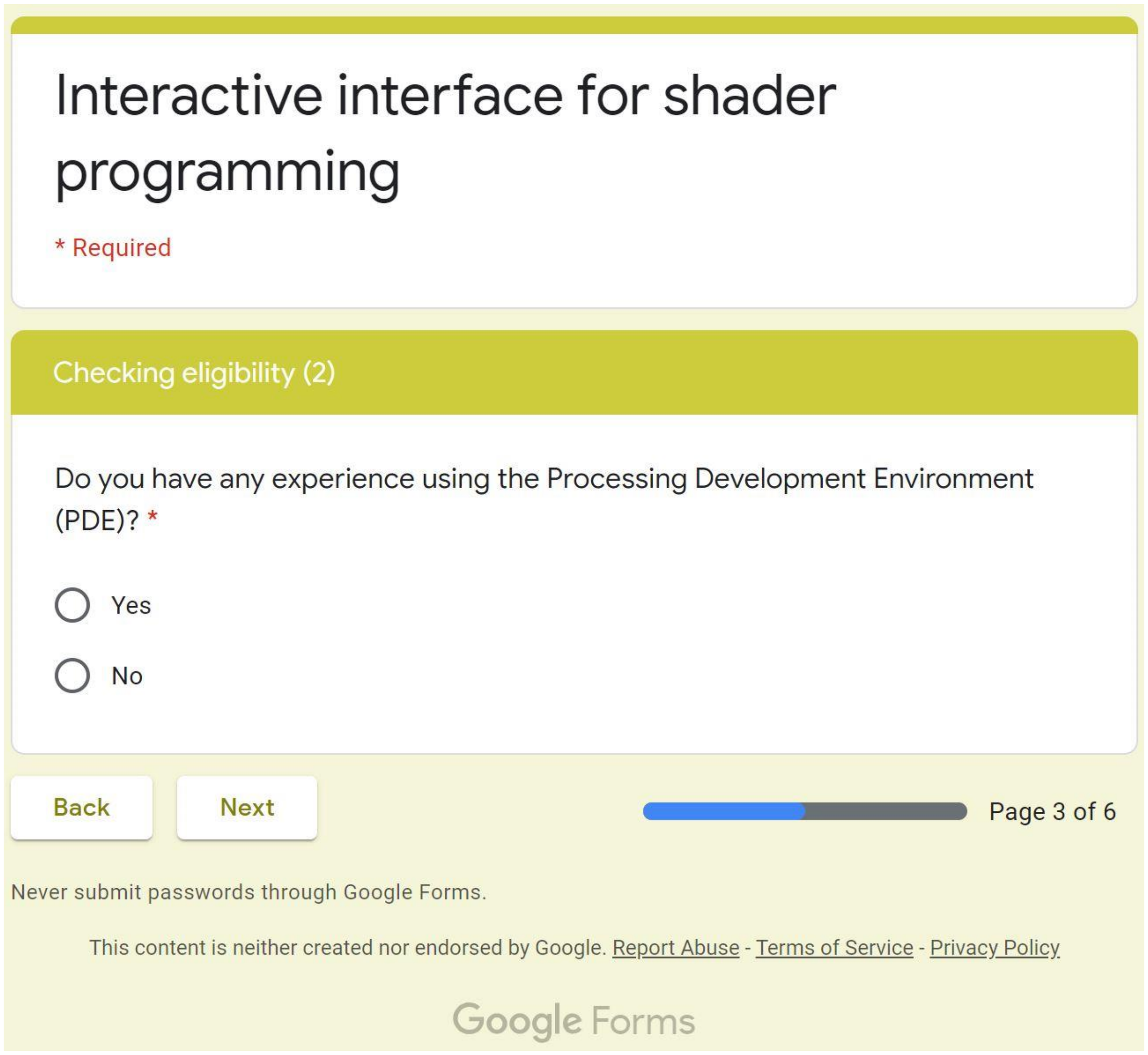
○ No

Back Next Page 2 of 6

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. Report Abuse - Terms of Service - Privacy Policy

Google Forms



Interactive interface for shader programming

* Required

Checking eligibility (2)

Do you have any experience using the Processing Development Environment (PDE)? *

○ Yes

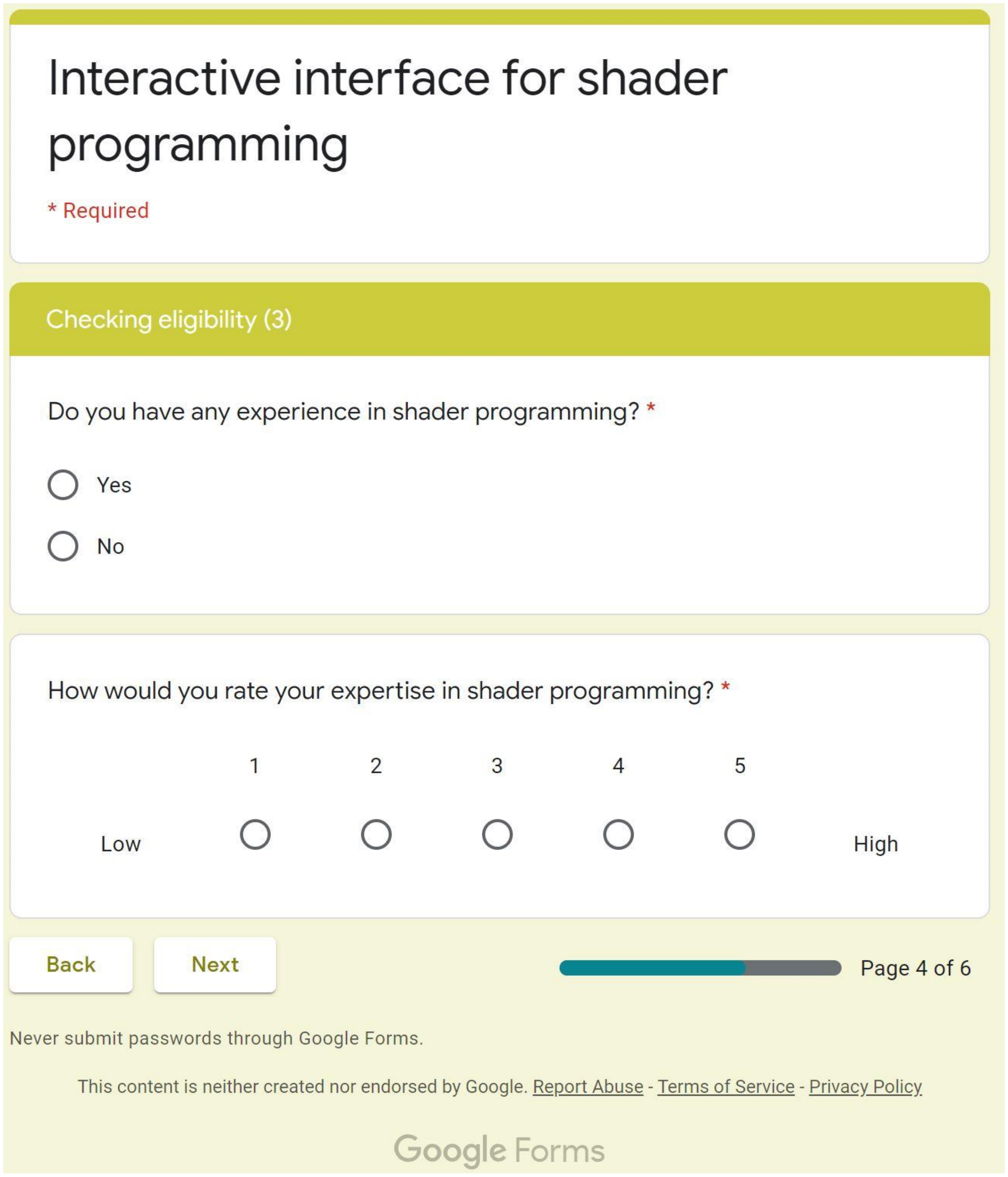
○ No

Back Next Page 3 of 6

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. Report Abuse - Terms of Service - Privacy Policy

Google Forms



Interactive interface for shader programming

* Required

Checking eligibility (3)

Do you have any experience in shader programming? *

○ Yes

○ No

How would you rate your expertise in shader programming? *

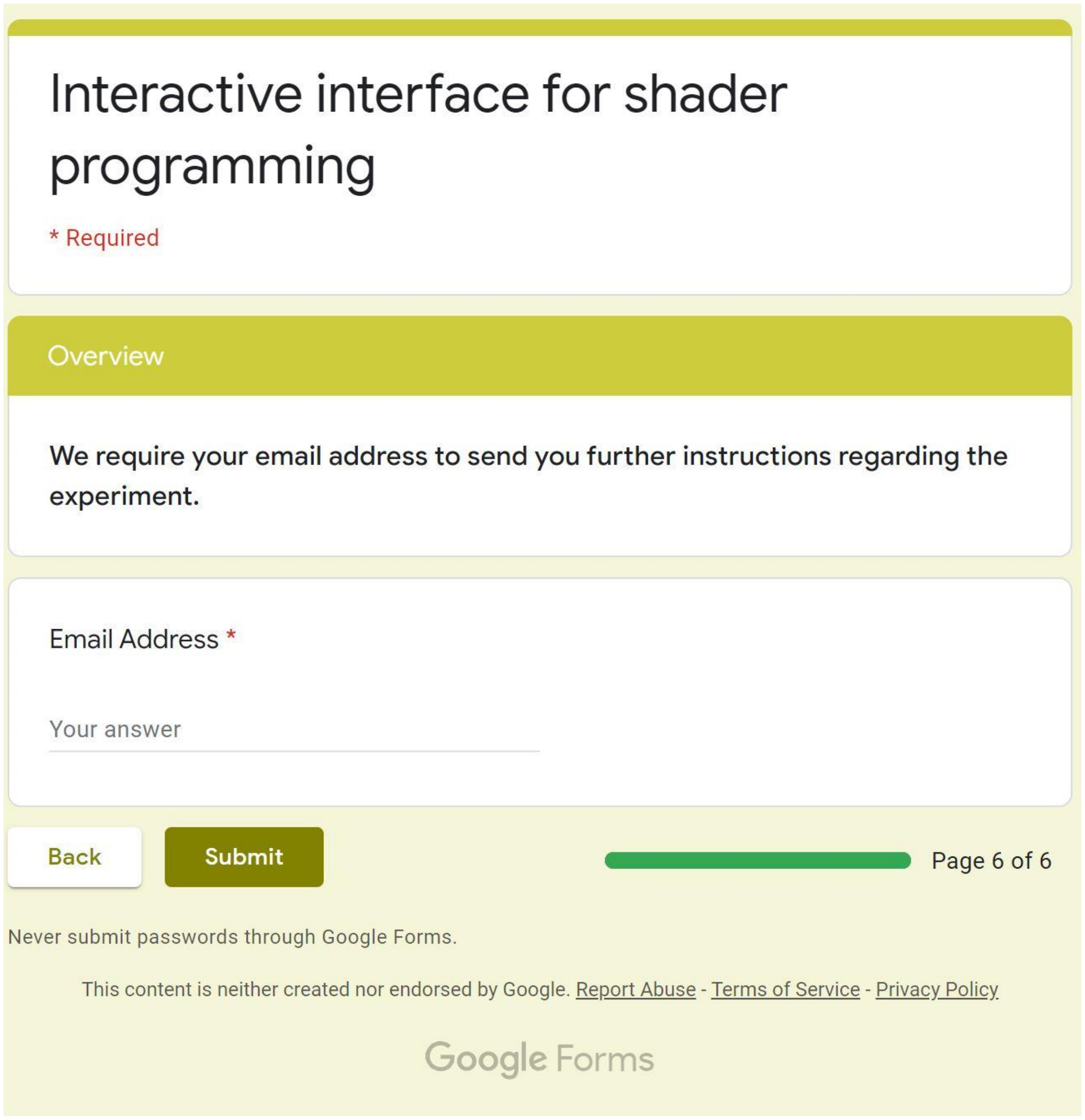
	1	2	3	4	5	
Low	○	○	○	○	○	High

Back Next Page 4 of 6

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. Report Abuse - Terms of Service - Privacy Policy

Google Forms



Interactive interface for shader programming

* Required

Overview

We require your email address to send you further instructions regarding the experiment.

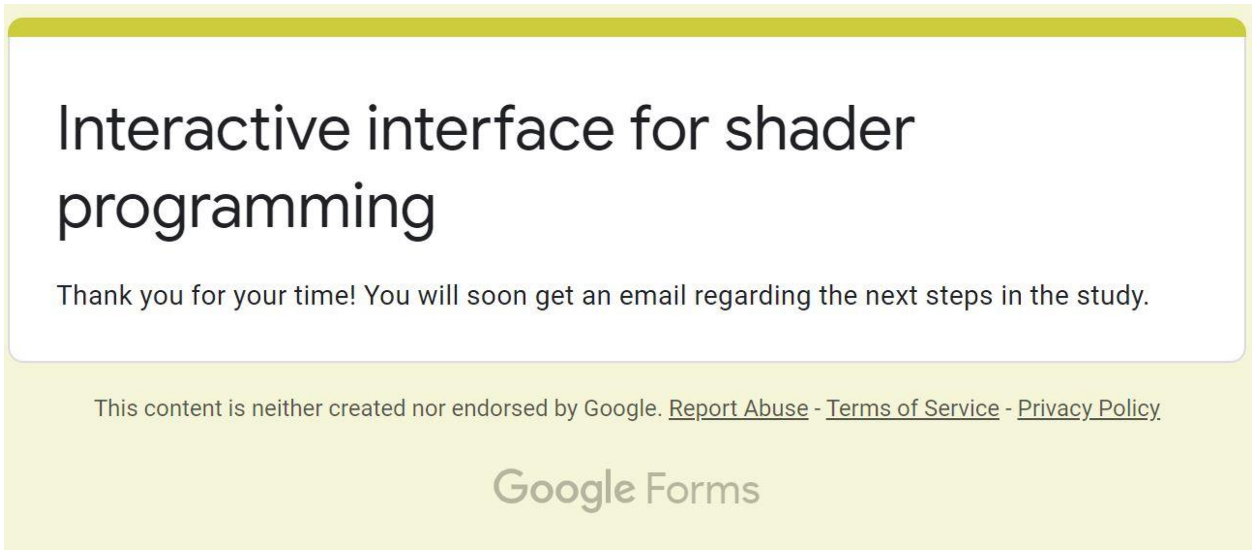
Email Address *

Your answer

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. Report Abuse - Terms of Service - Privacy Policy

Google Forms



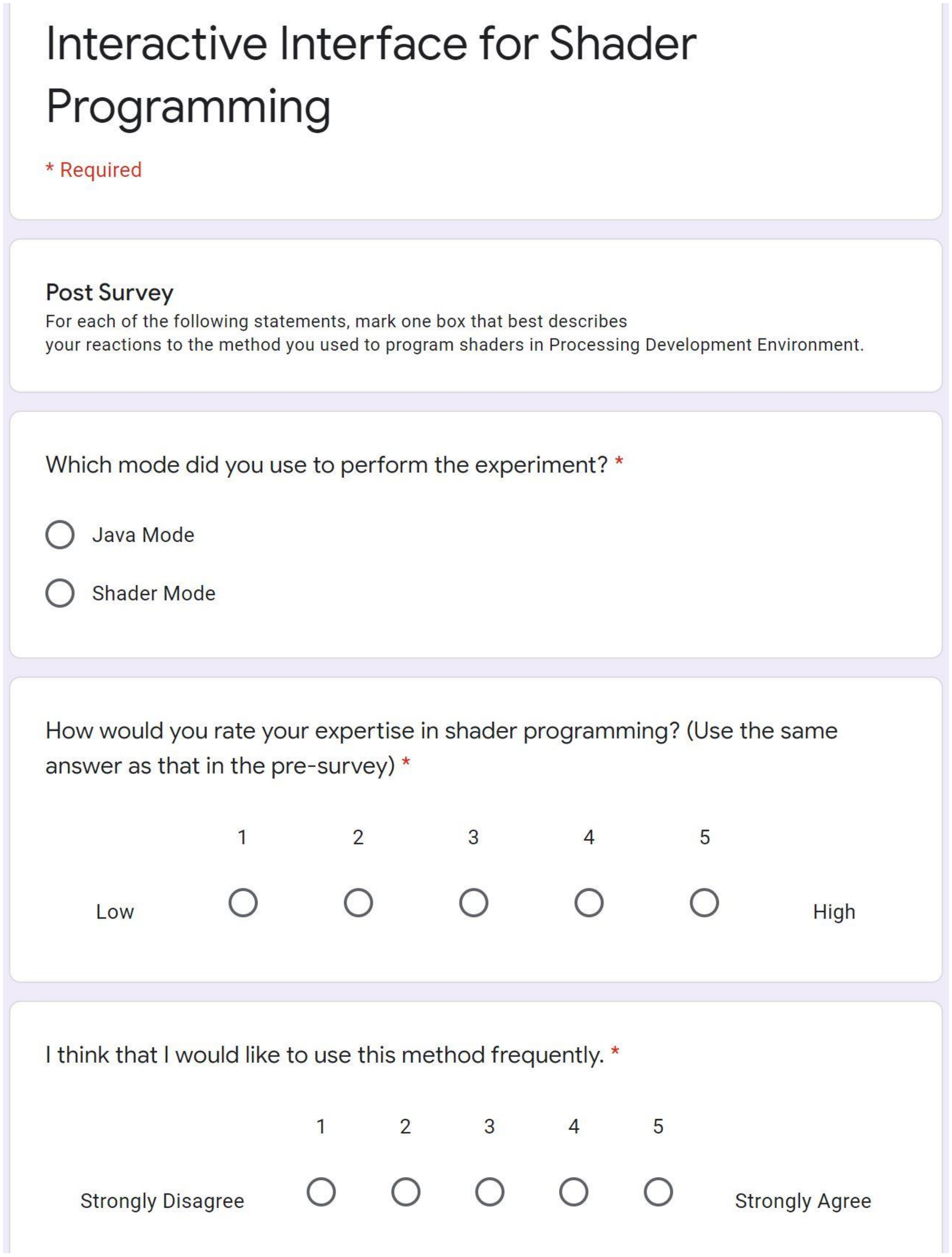
Interactive interface for shader programming

Thank you for your time! You will soon get an email regarding the next steps in the study.

This content is neither created nor endorsed by Google. Report Abuse - Terms of Service - Privacy Policy

Google Forms

Post-survey



Interactive Interface for Shader Programming

* Required

Post Survey

For each of the following statements, mark one box that best describes
your reactions to the method you used to program shaders in Processing Development Environment.

Which mode did you use to perform the experiment? *

○ Java Mode

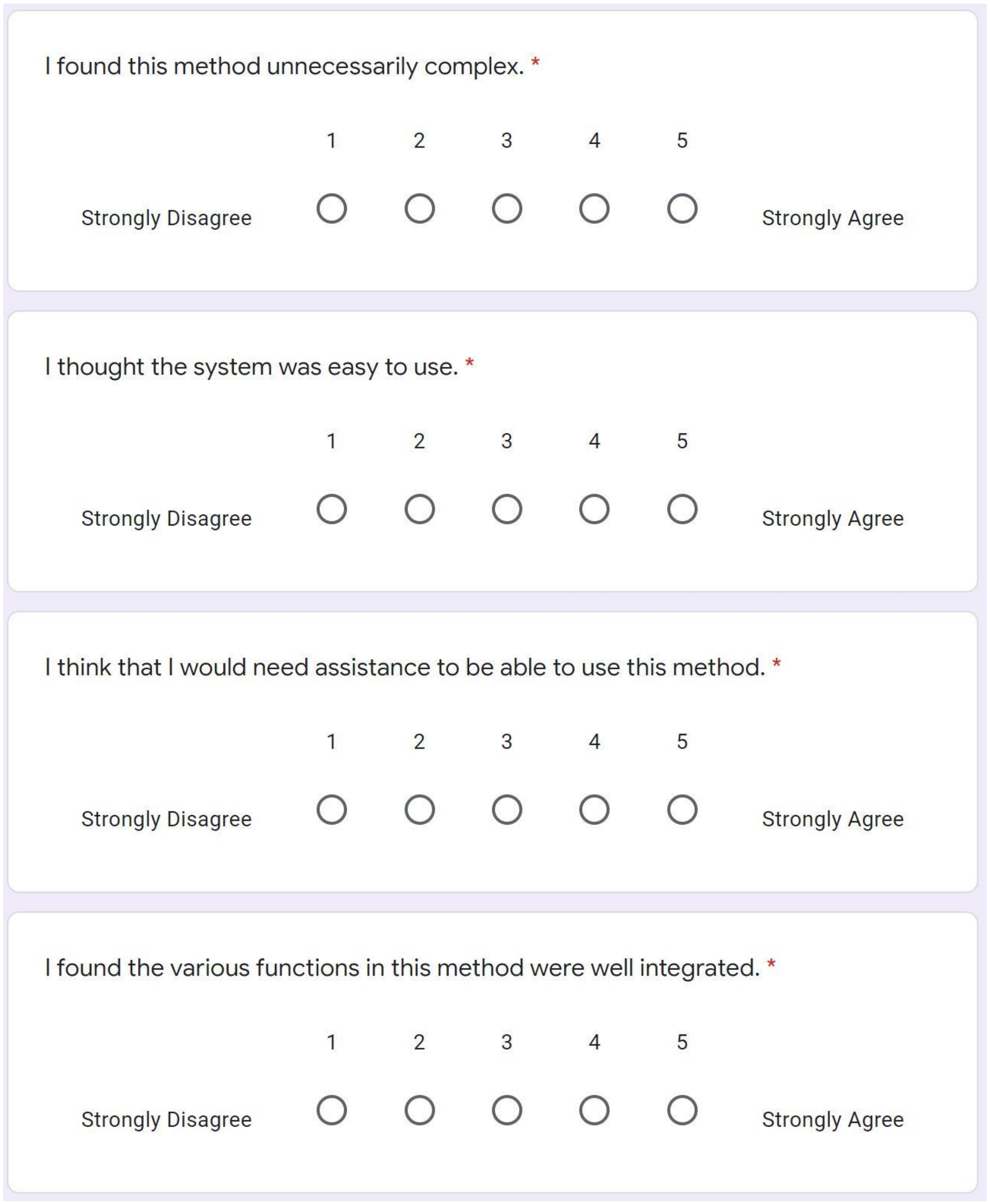
○ Shader Mode

How would you rate your expertise in shader programming? (Use the same answer as that in the pre-survey) *

	1	2	3	4	5	
Low	○	○	○	○	○	High

I think that I would like to use this method frequently. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree



I found this method unnecessarily complex. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I thought the system was easy to use. *

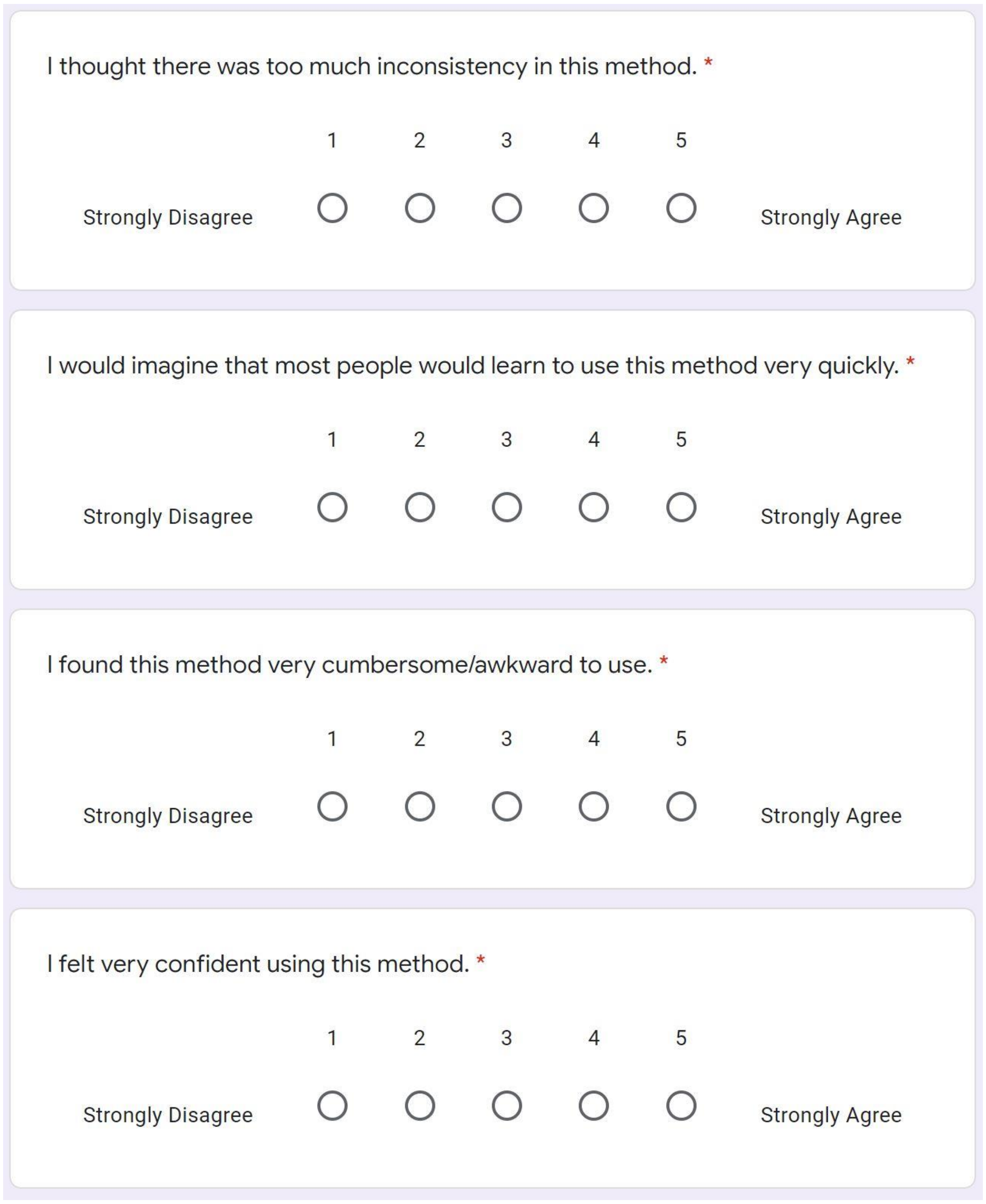
	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I think that I would need assistance to be able to use this method. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I found the various functions in this method were well integrated. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree



I thought there was too much inconsistency in this method. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I would imagine that most people would learn to use this method very quickly. *

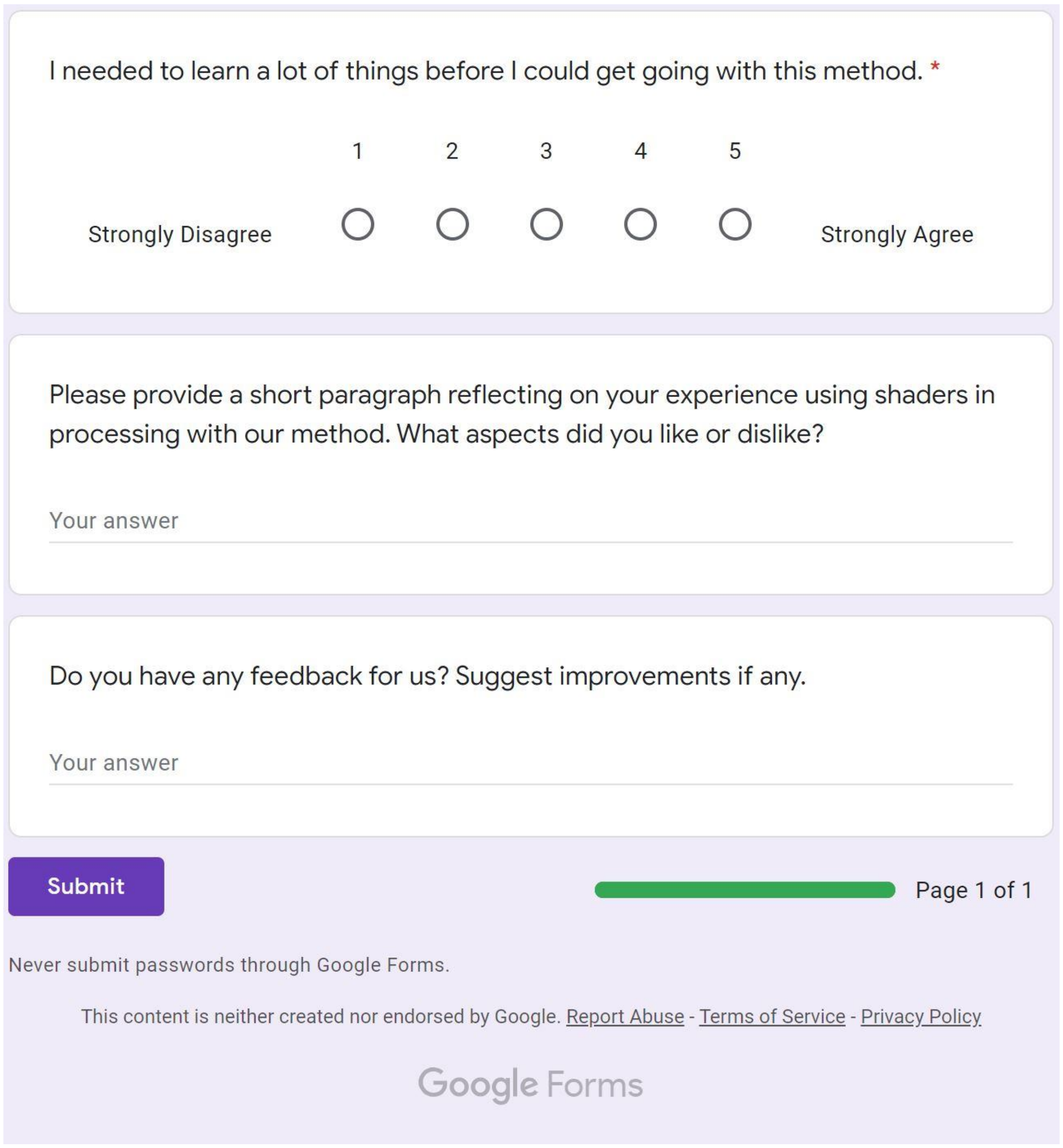
	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I found this method very cumbersome/awkward to use. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I felt very confident using this method. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree



I needed to learn a lot of things before I could get going with this method. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Please provide a short paragraph reflecting on your experience using shaders in processing with our method. What aspects did you like or dislike?

Your answer

Do you have any feedback for us? Suggest improvements if any.

Your answer

Submit

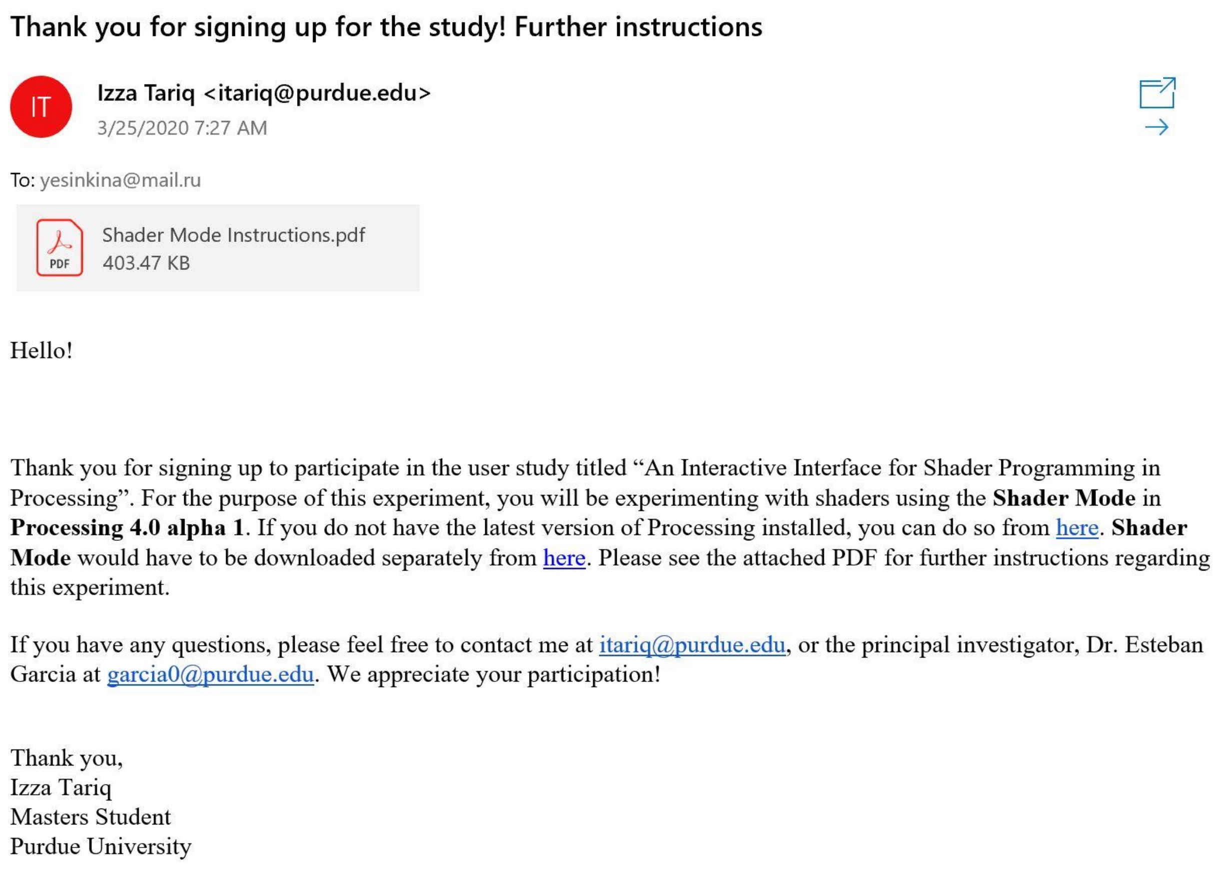
Page 1 of 1

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. Report Abuse - Terms of Service - Privacy Policy

Google Forms

APPENDIX C. EMAIL



Thank you for signing up for the study! Further instructions

IT

Izza Tariq <itariq@purdue.edu>
3/25/2020 7:27 AM

To: yesinkina@mail.ru

Shader Mode Instructions.pdf
403.47 KB

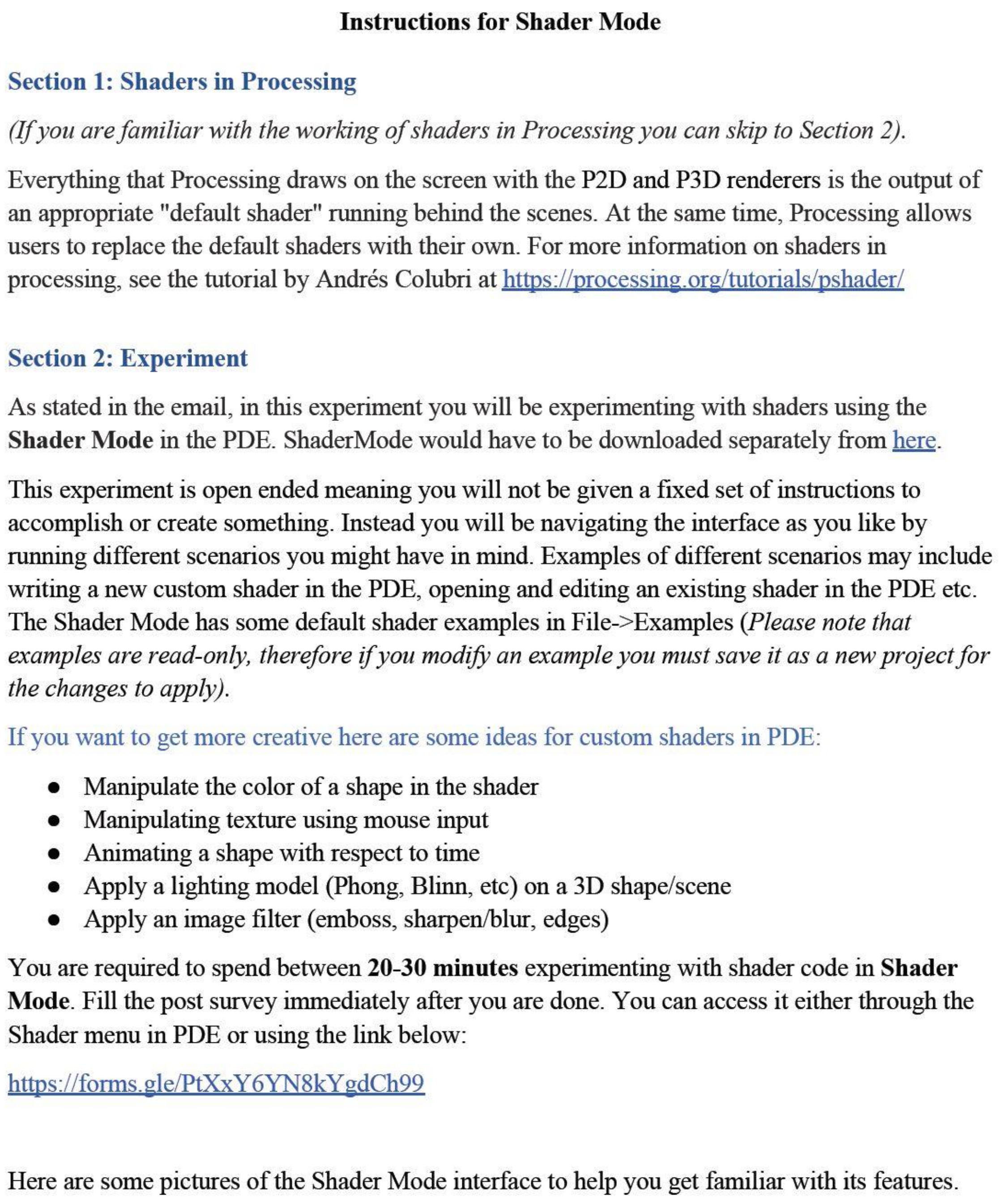
Hello!

Thank you for signing up to participate in the user study titled "An Interactive Interface for Shader Programming in Processing". For the purpose of this experiment, you will be experimenting with shaders using the **Shader Mode** in **Processing 4.0 alpha 1**. If you do not have the latest version of Processing installed, you can do so from here. **Shader Mode** would have to be downloaded separately from here. Please see the attached PDF for further instructions regarding this experiment.

If you have any questions, please feel free to contact me at itariq@purdue.edu, or the principal investigator, Dr. Esteban Garcia at garcia0@purdue.edu. We appreciate your participation!

Thank you,
Izza Tariq
Masters Student
Purdue University

APPENDIX D. SHADER MODE INSTRUCTIONS



Instructions for Shader Mode

Section 1: Shaders in Processing

(If you are familiar with the working of shaders in Processing you can skip to Section 2).

Everything that Processing draws on the screen with the P2D and P3D renderers is the output of an appropriate "default shader" running behind the scenes. At the same time, Processing allows users to replace the default shaders with their own. For more information on shaders in processing, see the tutorial by Andrés Colubri at https://processing.org/tutorials/pshader/

Section 2: Experiment

As stated in the email, in this experiment you will be experimenting with shaders using the **Shader Mode** in the PDE. ShaderMode would have to be downloaded separately from here.

This experiment is open ended meaning you will not be given a fixed set of instructions to accomplish or create something. Instead you will be navigating the interface as you like by running different scenarios you might have in mind. Examples of different scenarios may include writing a new custom shader in the PDE, opening and editing an existing shader in the PDE etc. The Shader Mode has some default shader examples in File->Examples (*Please note that examples are read-only, therefore if you modify an example you must save it as a new project for the changes to apply)*.

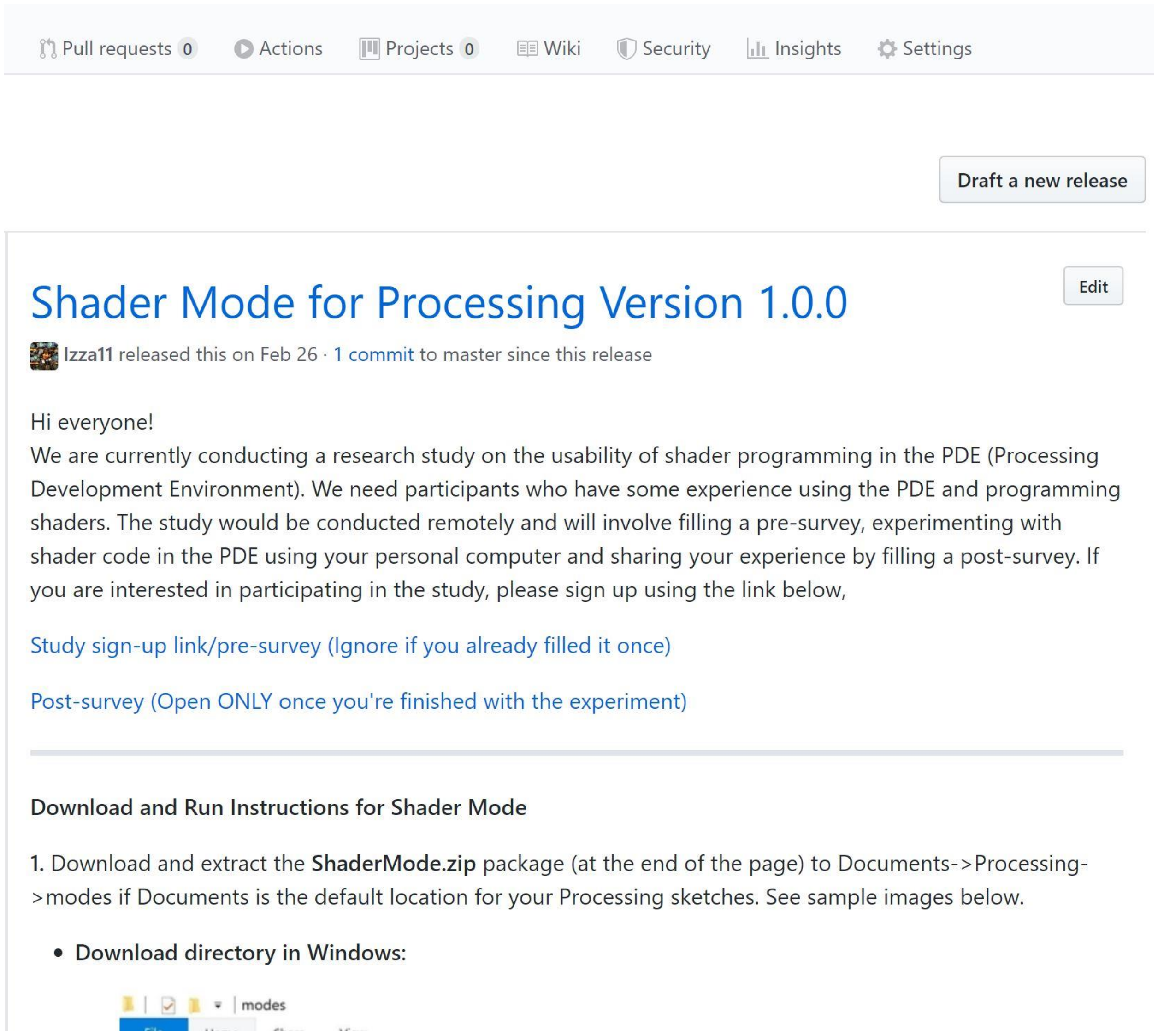
If you want to get more creative here are some ideas for custom shaders in PDE:

- Manipulate the color of a shape in the shader
- Manipulating texture using mouse input
- Animating a shape with respect to time
- Apply a lighting model (Phong, Blinn, etc) on a 3D shape/scene
- Apply an image filter (emboss, sharpen/blur, edges)

You are required to spend between **20-30 minutes** experimenting with shader code in **Shader Mode**. Fill the post survey immediately after you are done. You can access it either through the Shader menu in PDE or using the link below:

https://forms.gle/PtXxY6YN8kYgdCh99

Here are some pictures of the Shader Mode interface to help you get familiar with its features.



Shader Mode for Processing Version 1.0.0

Izza11 released this on Feb 26 · 1 commit to master since this release

Hi everyone!
We are currently conducting a research study on the usability of shader programming in the PDE (Processing Development Environment). We need participants who have some experience using the PDE and programming shaders. The study would be conducted remotely and will involve filling a pre-survey, experimenting with shader code in the PDE using your personal computer and sharing your experience by filling a post-survey. If you are interested in participating in the study, please sign up using the link below,

Study sign-up link/pre-survey (Ignore if you already filled it once)

Post-survey (Open ONLY once you're finished with the experiment)

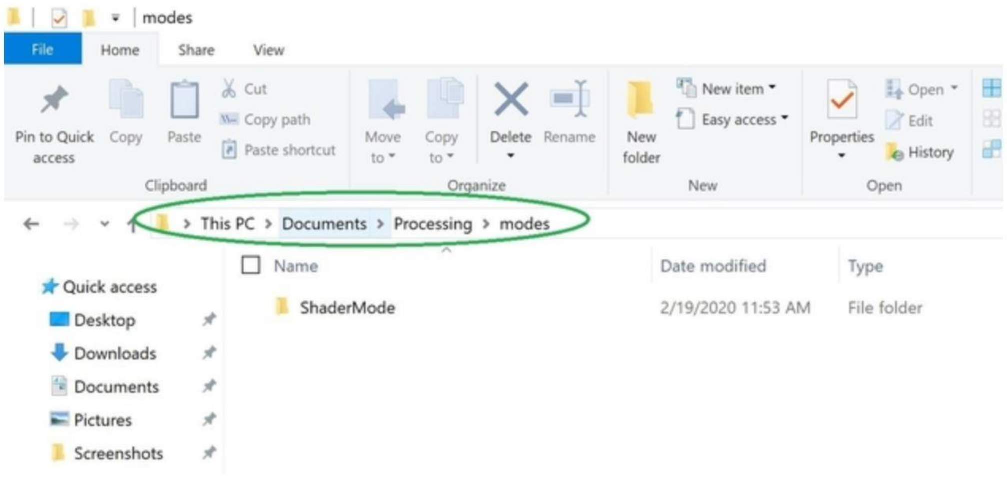
Download and Run Instructions for Shader Mode

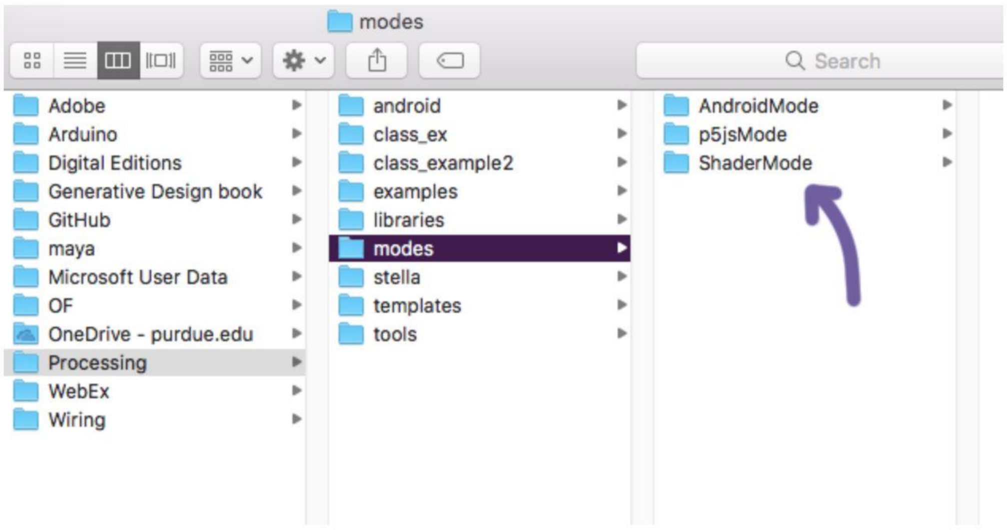
1. Download and extract the **ShaderMode.zip** package (at the end of the page) to Documents->Processing->modes if Documents is the default location for your Processing sketches. See sample images below.

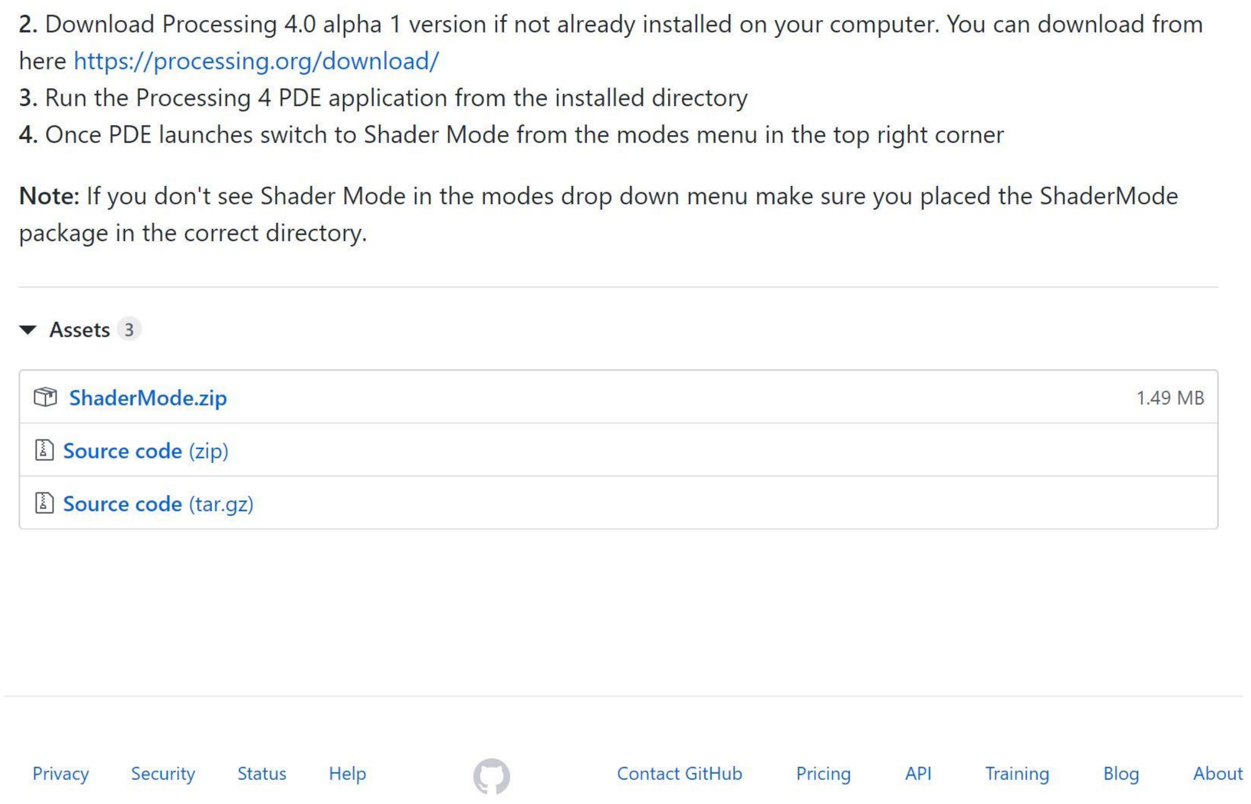
- **Download directory in Windows:**

Download and Run Instructions for Shader Mode

1. Download and extract the **ShaderMode.zip** package (at the end of the page) to Documents->Processing->modes if Documents is the default location for your Processing sketches. See sample images below.

- **Download directory in Windows:**
    - 

- **Download directory in Mac:**
    - 



2. Download Processing 4.0 alpha 1 version if not already installed on your computer. You can download from here https://processing.org/download/
3. Run the Processing 4 PDE application from the installed directory
4. Once PDE launches switch to Shader Mode from the modes menu in the top right corner

Note: If you don't see Shader Mode in the modes drop down menu make sure you placed the ShaderMode package in the correct directory.

REFERENCES

A sophisticated text editor for code, markup and prose. (n.d.). Retrieved from https://www.sublimetext.com/

Affairs, A. S. (2013, September 06). System Usability Scale (SUS). Retrieved from https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

Apodaca, A., & Mantle, M. (1990). RenderMan: Pursuing the future of graphics. *Computer Graphics and Applications, IEEE, 10*(4), 44-49.

Bailey, M. (2007). Teaching OpenGL shaders: Hands-on, interactive, and immediate feedback. *Computers & Graphics,31*(3), 524-531. doi:10.1016/j.cag.2007.02.001

Bangor, A., Kortum, P. T., & Miller, J. T. (2008). An Empirical Evaluation of the System Usability Scale. *International Journal of Human-Computer Interaction*, *24*(6), 574–594. doi: 10.1080/10447310802205776

Binyamin, S., Rutter, M., & Smith, S. (2016). The Utilization Of System Usability Scale In Learning Management Systems: A Case Study Of Jeddah Community College. *ICERI2016 Proceedings*. doi: 10.21125/iceri.2016.2290

Brutzman, Don. (2015). X3D: Extensible 3D Graphics for Web Authors.

Colubri, A. (n.d.). Some shadertoy shaders run slower on Processing · Issue #1714 · processing/processing. Retrieved from https://github.com/processing/processing/issues/1714

Cabello, R. (n.d.). GLSL Sandbox. Retrieved from http://glslsandbox.com/

Davis, J. (n.d.). Joshua Davis Studios. Retrieved from https://joshuadavis.com/

FoodRisc Resource Centre. (n.d.). Retrieved from http://resourcecentre.foodrisc.org/mixed-methods-research_185.html

Fosner, R. (2003). *Real-time shader programming covering DirectX 9.0* (The Morgan Kaufmann Series in Computer Graphics). San Francisco: Morgan Kaufmann.

Garcia, A. (2013, November 27). UX Research | Standardized Usability Questionnaire. Retrieved from https://arl.human.cornell.edu/linked docs/Choosing the Right Usability Questionnaire.pdf

Gómez, A. F., Colubri, A. P., & Charalambos, J. (2016). Shader programming for computational arts and design: A comparison between creative coding frameworks. *VISIGRAPP 2016 – Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications,* 163-170.

Gomez, A. F., Charalambos, J. P., & Colubri, A. (2016). ShaderBase: A Processing Tool for Shaders in Computational Arts and Design. *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. doi:10.5220/0005673201890194

Jensen, P., Francis, N., Larsen, B., & Christensen, N. (2007). Interactive shader development. *Proceedings of the 2007 ACM SIGGRAPH Symposium on Video Games*, 89-95.

Jeremias, P., & Quilez, I. (2014). Shadertoy. *SIGGRAPH Asia 2014 Courses on - SA 14*. doi:10.1145/2659467.2659474

KodeLife. (n.d.). Retrieved from https://hexler.net/software/kodelife/

Kolaczynski, K. (n.d.). Synthclipse – Home. Retrieved from http://synthclipse.sourceforge.net/

Kwok, R. (n.d.). Retrieved from http://ravenkwok.com/

Luebke, D., & Humphreys, G. (2007). How GPUs Work. *Computer,40*(2), 96-100. doi:10.1109/mc.2007.59

Maughan, C., & Wloka, M. (2001). Vertex shader introduction. *NVIDIA Technical Brief*.

Ryan, G. W., & Bernard, H. R. (2000). Techniques to identify themes in qualitative data. *Handbook of Qualitative Research. 2nd ed. Thousand Oaks, CA: Sage Publications*.

Sauro, J. (2011, February 02). Measuring Usability with the System Usability Scale (SUS). Retrieved from https://measuringu.com/sus/

Sauro, J. (2016, May 02). Measuring Usability With The System Usability Scale (SUS). Retrieved from https://www.userfocus.co.uk/articles/measuring-usability-with-the SUS.html

OpenGL Shading Language. (n.d.). Retrieved from https://www.khronos.org/opengl/wiki/OpenGL_Shading_Language

Ousterhout, J. K. (1998). Scripting: Higher level programming for the 21st Century. *IEE Computer,31*(3), 23-30. doi:10.1109/2.660187

Owen, G., Zhu, Y., Chastine, J., & Payne, B. (2005). Teaching programmable shaders:lightweight versus heavyweight approach. International Conference on Computer Graphics and Interactive Techniques: ACM SIGGRAPH 2005 Educators Program : Los Angeles, California; 31 July-04 Aug. 2005.

Pazos, A. (n.d.). hamoid - Overview. Retrieved from https://github.com/hamoid

Processing Foundation. (n.d.). Retrieved from https://processing.org/

Papagiannakis, G., Papanikoalou, P., Greassidou, E., & Trahanias. (2014). GlGA: An OpenGL Geometric Application framework for a modern, shader-based computer graphics curriculum. Computer Graphics Forum, 33(2), 11-16.

Quilez, I. (n.d.). Shadertoy BETA. Retrieved from https://www.shadertoy.com/

Reas, C., & Fry, B. (2014). *Processing: A programming handbook for visual designers and artists*. Cambridge, MA: The MIT Press.

Reas, C., & Fry, B. (2003). Processing a learning environment for creating interactive Web graphics. ACM SIGGRAPH 2003 Web Graphics, SIGGRAPH 2003.

RenderMonkey Toolsuite - IDE Features. (n.d.). Retrieved from https://gpuopen.com/archive/gamescgi/rendermonkey-toolsuite/rendermonkey-toolsuite ide-features/

Sarkar, A. (2015). The impact of syntax colouring on program comprehension. *Proceedings of the 26th Annual Conference of the Psychology of Programming Interest Group*, 1–10.

Satran, M. (n.d.). HLSL. Retrieved from https://docs.microsoft.com/en us/windows/desktop/direct3dhlsl/dx-graphics-hlsl

SUS: The System Usability Scale. (n.d.). Retrieved from https://www.trymyui.com/sus-system-usability-scale

Tanimoto, S. (2013). A perspective on the evolution of live programming. Live Programming (LIVE), 2013 1st International Workshop on, 31-34.

Tariq, I. (2018, August 14). Final GSOC Report. Retrieved from https://www.izzatariq.com/final-gsoc-report

Tatarchuk, N. (2004). RenderMonkey: An effective environment for shader prototyping and development. *ACM SIGGRAPH 2004 Sketches*, 91.

Vivo, P. G., & Lowe, J. (n.d.). The Book of Shaders. Retrieved December 05, 2018, from https://thebookofshaders.com/