Graphite: Open Source 2D Graphics Editor

A Senior Project
presented to
the Faculty of the Computer Science Department
California Polytechnic State University – San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Science

By

Oliver Dolan, Christopher Mendoza

June 11, 2023

Website: https://editor.graphite.rs/

Introduction

Graphite is an open source 2D graphics editor aiming to provide a useful alternative to the Adobe product suite while integrating modern ideas from research and industry, including a node-based procedural approach that makes the design process fully non-destructive. Tools from the VFX and gamedev industry like Nuke, Houdini, and Substance Designer have proven the power and flexibility of node-based systems, but they are each lacking in their user experience because of the deep level of complexity that is not abstracted into simpler concepts for users (Adobe Systems, SideFX, The Foundry Visionmongers Ltd., n.d.). Graphite puts the node-based core into a traditional tool-based shell, making it more accessible and familiar to experienced 2D designers and artists as well as new users. These tools, which form an abstraction around the node graph concepts, act much like existing graphics editors. One aspect of tool-based editing is the snapping system that can constrain artwork manipulations to align with other layers, geometry, or a grid. Graphite seeks to improve upon the user experience of many core tools and workflows in areas neglected by traditional editing software, and snapping systems are one prime example of a common, “boring” feature where pain points and potential improvements are hidden in plain sight. Graphite’s goal is to fundamentally improve upon the user experience of snapping so artists and designers can benefit from a more useful way to make pixel-perfect artwork.

Background:

Both of us had an interest in working on a graphics application. Applications like Adobe illustrator, and other graphics editors have widespread usage in both professional work and personal creative pursuits. We had both had experience using similar software for game development assets and in frontend development. This motivated us to join the Graphite team for our senior project and contribute to the development process of a graphics editor. We thought it would be interesting to learn about ways people approach coding these types of applications.

Related Work

As previously mentioned our main related work consists of the Adobe product suite, as well as VFX tools like Nuke and Houdini. We most prominently base our work off the functionality of Adobe products like Illustrator and Photoshop, on first glance our interface will look very similar to Illustrator as we wanted to stay consistent with what our audience was used to in their UI for editing tools. We differ from Adobe Illustrator with our node based system as that was one of the main motivations of Graphite, to innovate the Adobe suite with modern design principles.

With the VFX tools Nuke and Houdini we are similar in that we aim to make use of node-based editing systems like their softwares do. We differ from Nuke and Houdini as they aim towards 3D audiences in photo and video editing while we focus on a 2D approach to image editing and don't include video functionality (Adobe Systems, SideFX, The Foundry Visionmongers Ltd., n.d.).

Design

Goals of work:
Improving Graphite's rudimentary snapping system (and implementing the non-existent grid snapping system) to become at least as good as other software on the market, and then using that as a testbed to experiment with new ways to make the UX as useful as possible under varied use cases.

Principles guiding implementation:
This is the fundamental design principles we follow with Graphite:

→ Understand the problem and what's desired
→ Explore the existing code and find similar already implemented examples to reference
→ Build an MVP
→ Get code quality/code approach feedback from code review
→ Polish the experience
→ Get feedback
→ Loop these last two steps
→ Merge

What sets us apart:
Graphite is aiming to become a full professional-grade graphics editor where no wishlist item (such as a superb grid/snapping system UX) is out of scope for being too ambitious. Existing snapping systems fail to achieve certain use cases like the tangent on a circle or curve, prioritize the snapping targets that are most likely to be useful, and provide hotkeys to manually engage certain desired snapping targets. These developments may also lead into the defining of constraints for a CAD-like constraint system (likely out of scope for this senior project).

Implementation

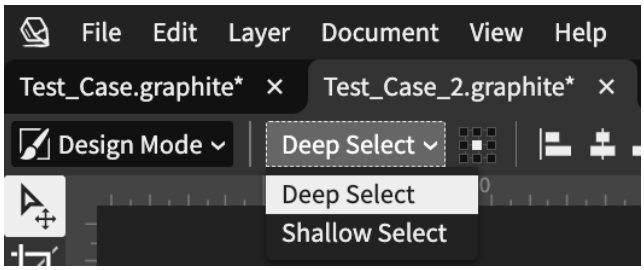
1. We used Rust in the Graphite codebase, the language of the future, because it makes it pleasant to implement highly scalable code bases, helps avoid bugs and ensure reliability, plays very nicely with the WebAssembly ecosystem used for the backend component of the web-based UI, a niche but growing community of really smart, driven developers eager to contribute to open source projects, an ecosystem of libraries that provide the perfect solutions for the tech stack
2. As for that tech stack, what we're using that's relevant to our tasks is a library called "glam", a linear algebra math library for vectors and matrices. For Graphite as a whole, wasm-bindgen for the WebAssembly target and Tauri for wrapping the web UI in a desktop app while running Rust code natively. We also borrowed necessary math functions to solve things like quartic functions for our tangent snapping issue.
3. Here are the github issues we've work worked throughout the winter and spring quarter:

First Quarter

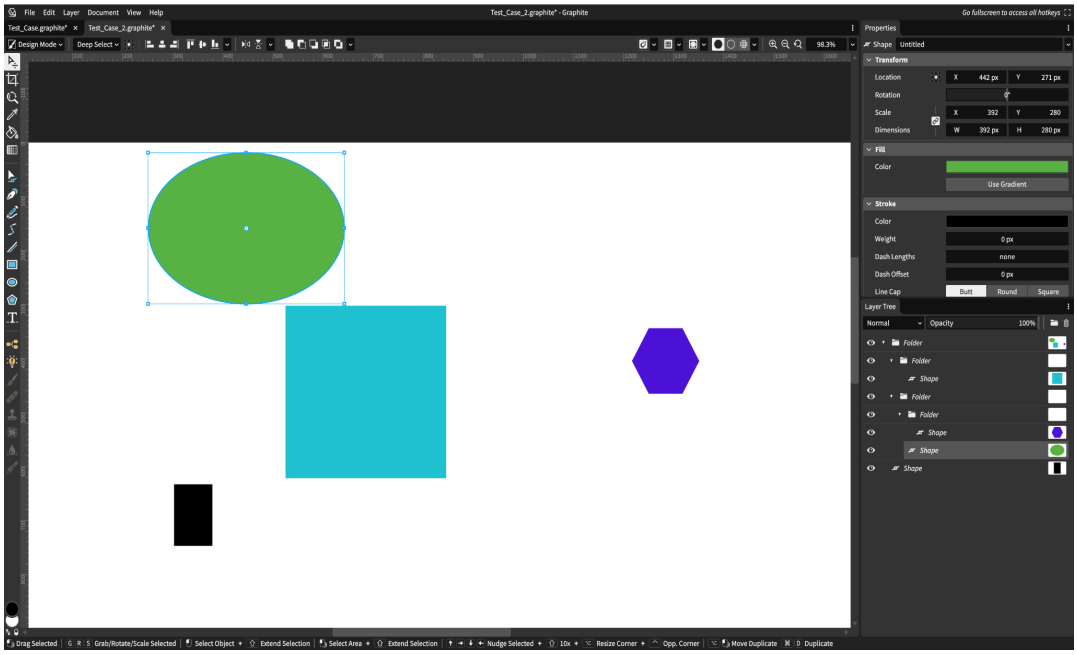
Improve the Select Tool with Deepest and Shallowest mode #985:
https://github.com/GraphiteEditor/Graphite/issues/985
Authors: Chris and Ollie
Overview: Add the ability to manipulate folders with the select tool
Work: We created a Radio button option for toggling between selecting Groups of layers (Shallowest Select) or an Individual layer (Deepest Select) for the Select Tool. This allows the user to have the ability to select and drag an individual shape or several shapes nested in folders. The behavior for selecting groups of shapes mimic the behavior of Adobe XD. Figuring out how to make our logic the same as Adobe XDs' was the majority of the work. To do this, we had to create identical test scenarios in both Graphite and Adobe XD and make sure that the selected layers were the same after each selection of another shape (Adobe Systems, n.d.). There were numerous edge cases to cover when calculating the next layer path. Some examples include, but not limited too:

- When multiple layers are currently selected, the new selected layer path should be the common path all the selected layers share.
- When multiple layers are currently selected and one of the currently does not share paths, the layer tree needs to be updated when the user shift clicks on a layer that is shallower than the selected
- Command clicking should mimic the behavior of the deepest selection

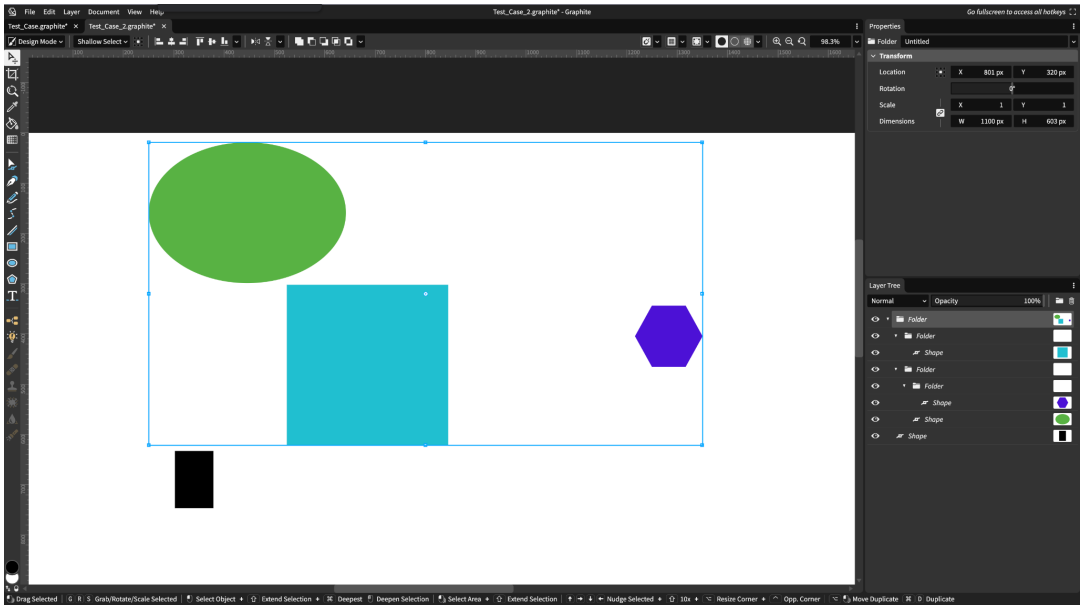
This code has been merged into the master branch.



Shallow Select/Deep Select Widget



Deepest Selection

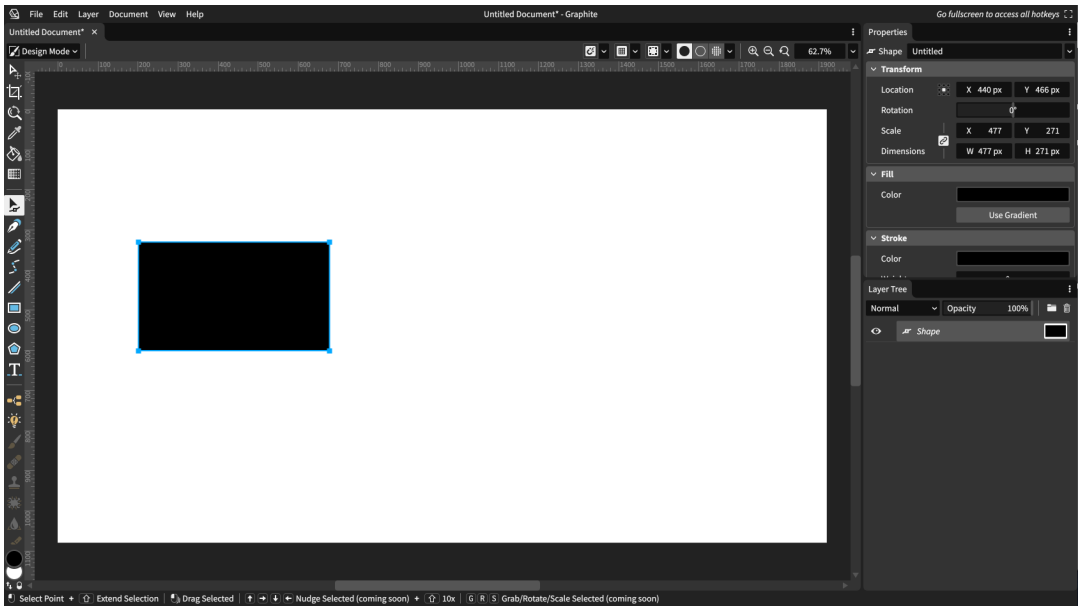


Shallowest Selection

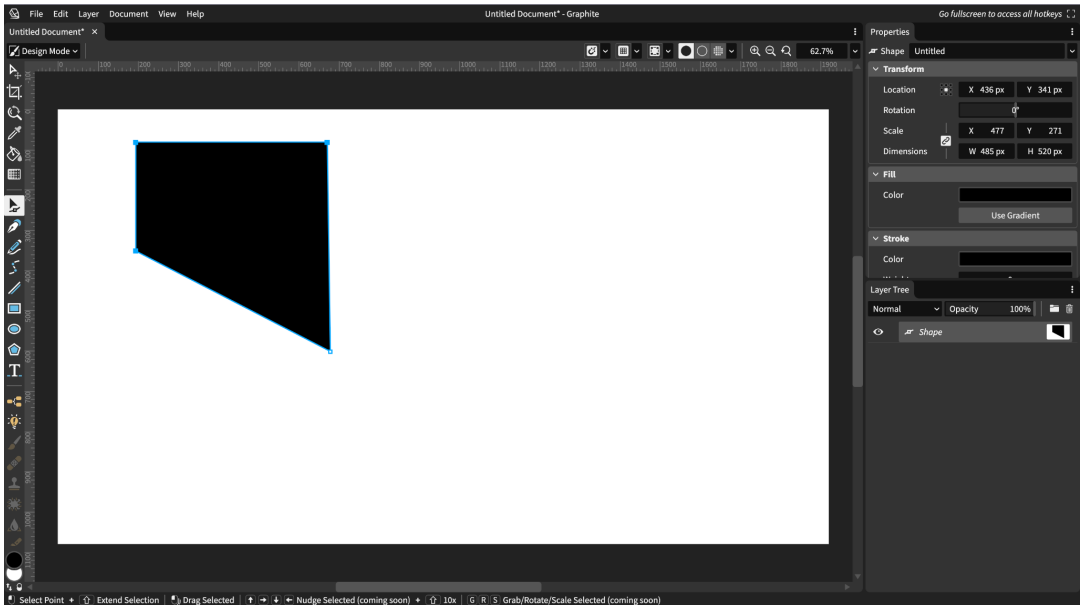
Playable at: https://editor.graphite.rs/

Improve the Path Tool and anchor point selection behavior #722:
https://github.com/GraphiteEditor/Graphite/issues/722
Authors: Chris and Ollie
Overview: Improve the selection and deselection of anchor points
Work: We added multiple improvements to the anchor point selection behavior. When clicking on a shape, all of the anchor points are now selected. In addition to this, we added the ability to

select and drag shapes similarly to the Select tool. We also worked on the behavior for shift clicking anchor points for selecting and deselecting anchor points. Lastly, we added some behavior improvements for the user clicking on an individual anchor point; it now deselects every other anchor point. This code has been merged into the master branch.



Selects all anchor points on first click



Shift click on the bottom right point, allows manipulation of other points

Playable at: https://editor.graphite.rs/

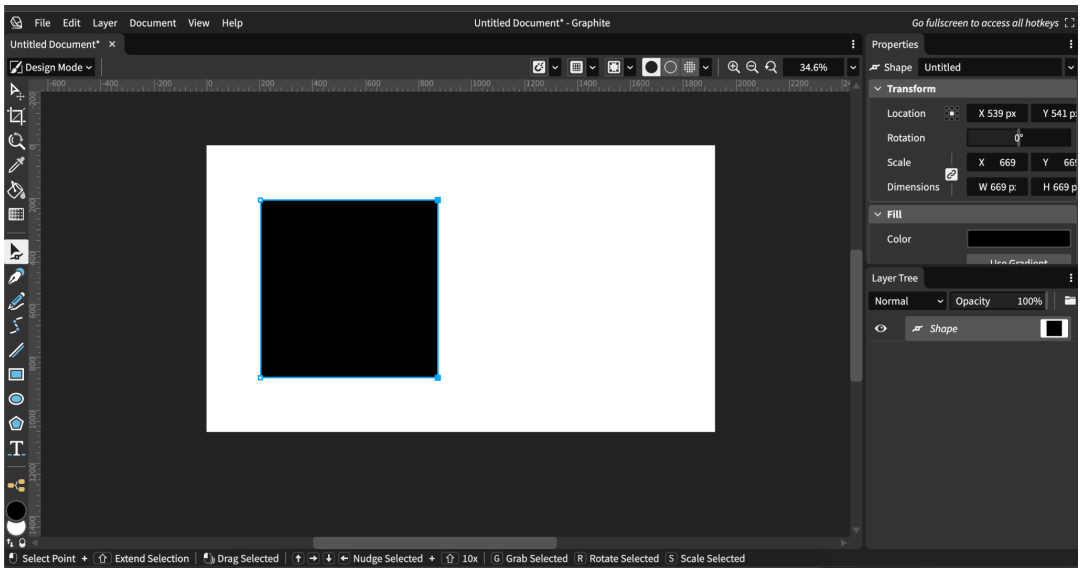
Allow nudging, angle snapping, and G/R/S transformation of selected points in the Path tool #820:

https://github.com/GraphiteEditor/Graphite/issues/820

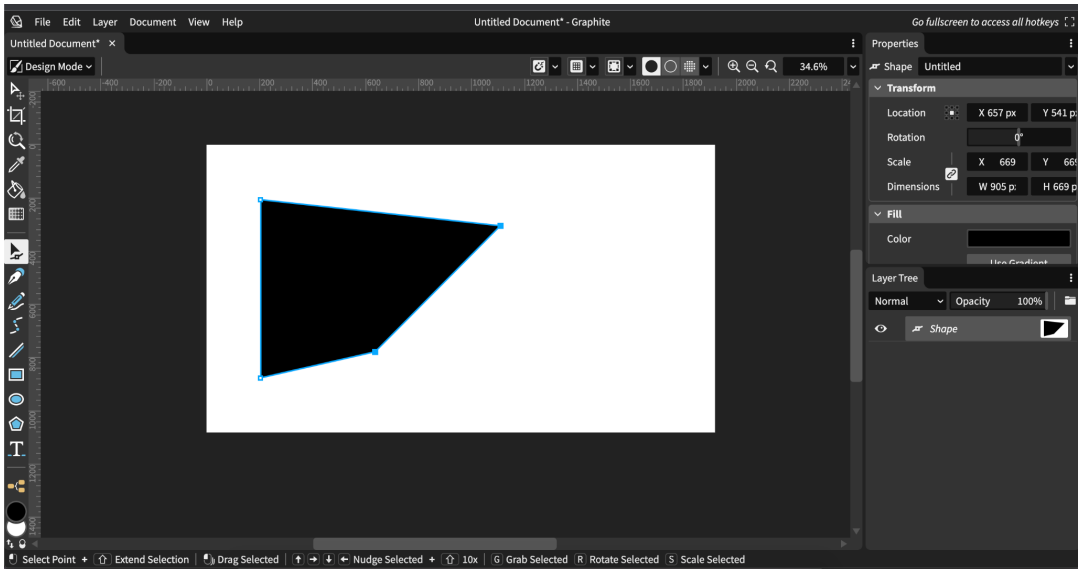
Authors: Ollie

Overview: The basis of this issue was to improve the point manipulation system, which entailed being able to nudge, rotate, grab, and scale selected points in the path tool.

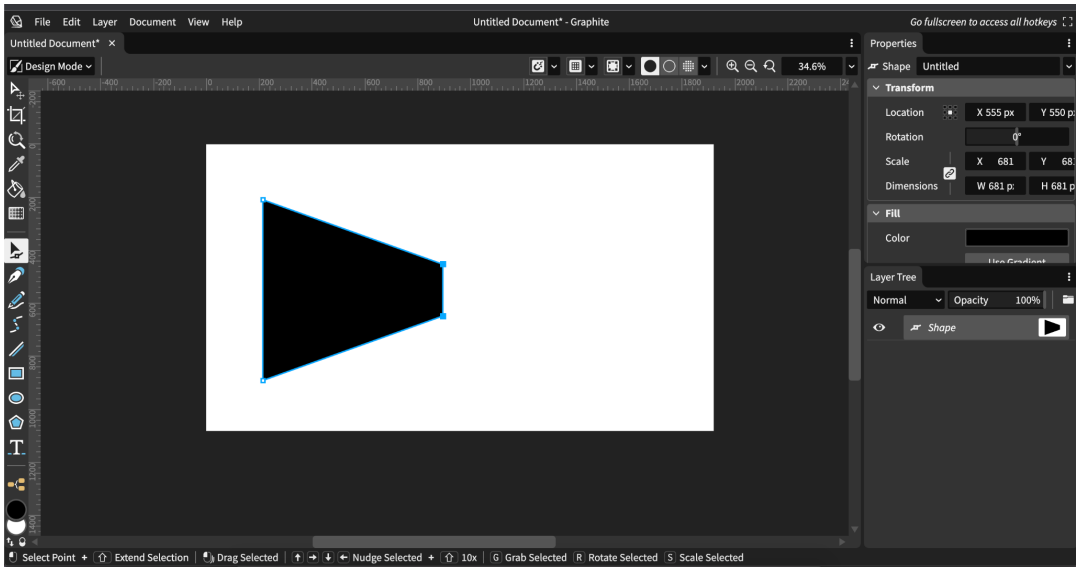
Work: Ollie worked on the rotate, grab, and scale element of this issue. This was difficult because I had to familiarize myself with a new part of the codebase to create this functionality. It initially took about 2 weeks to get it working, then it took another 2 weeks refactoring and adding the proper revisions from code review. My code is being reviewed now and hopefully is added to our working version once I've ironed out the last few kinks.



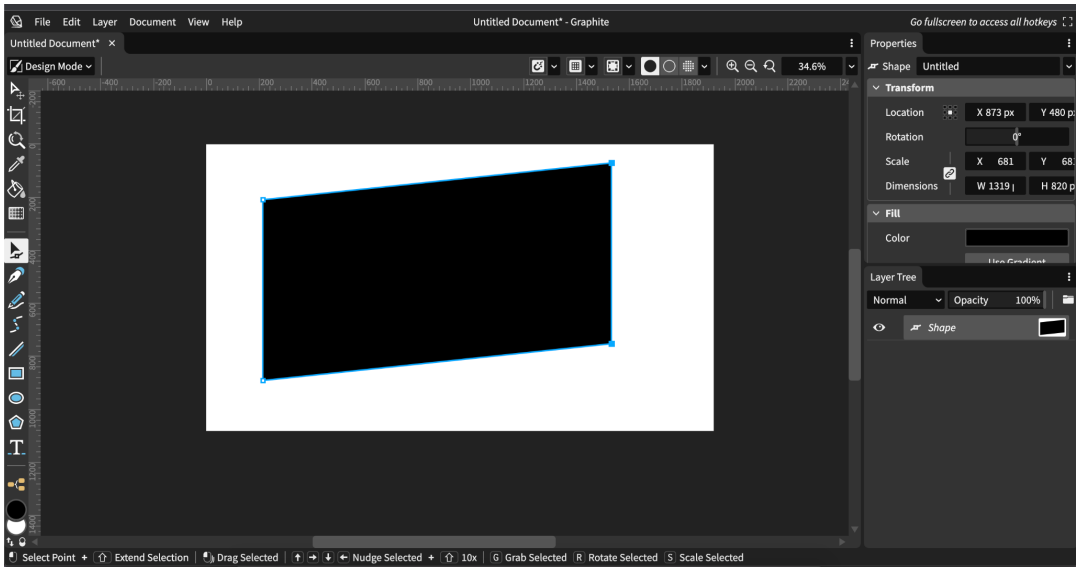
Original



Rotation of angle 45 degrees



Scaling the points



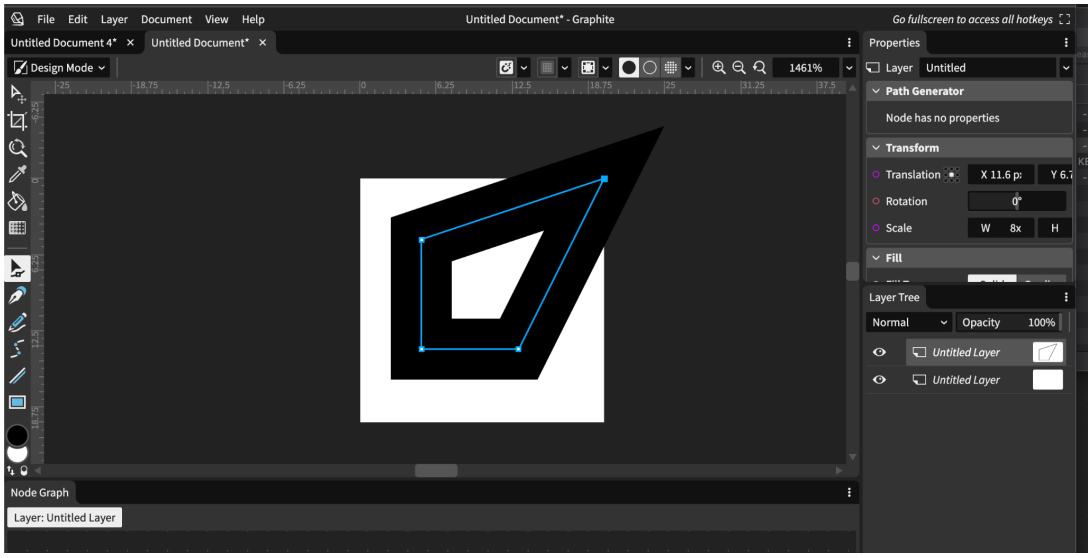
Grabbing the points

Playable at: https://editor.graphite.rs/

Second Quarter

Grid snapping system #318:

https://github.com/GraphiteEditor/Graphite/issues/318
Authors: Ollie and Chris
Overview: Transforming a layer or point, such as dragging, grabbing (the G of G/R/S), nudging, etc., should snap (round) it to the nearest integer in document coordinates
Work: The goal of this issue was to obviously round each point's position of a shape/layer to be a full integer in the **document space.** For context, document space is the pixel coordinates displayed on the canvas so for example in this screenshot, we are working with a 20x20 document space canvas (as you can see on the X and Y axis)



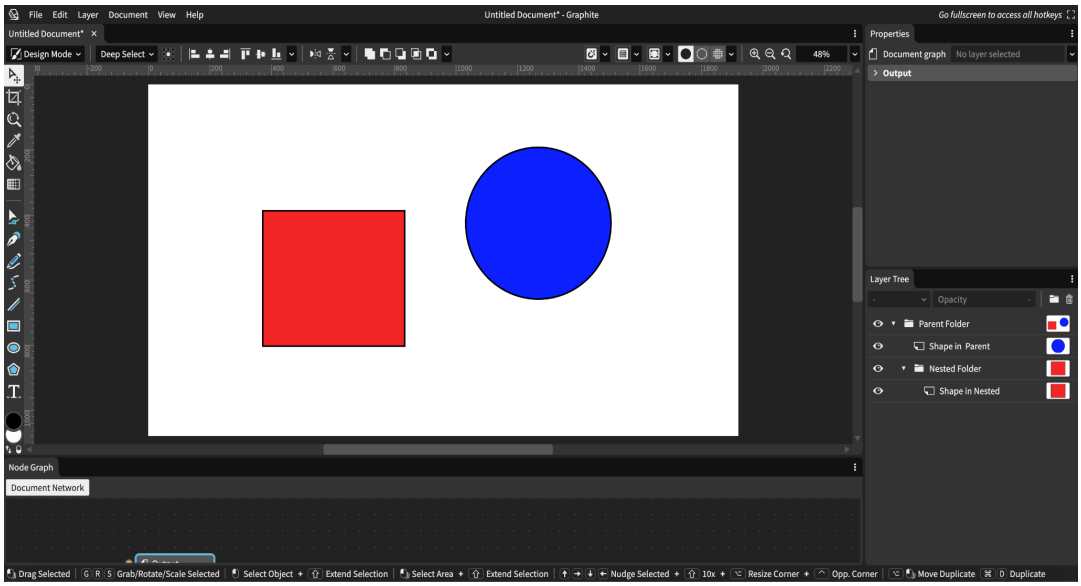
The main problem with this issue was translating the points position from **layer space**, which is a coordinate grid going from 0-1, meaning that if the shape was a square its points original positions are stored as (0,0) (0,1) (1,1) (1,0). Which then needs to be translated to viewport space which is determined based on the user's zoom in or out on the canvas, meaning that it's different from the document space because it changes whenever the user zooms. After translating from layer space to viewport space to document space, the majority of the issue was solved through that.

Playable at: https://d4e61d10.graphite.pages.dev/

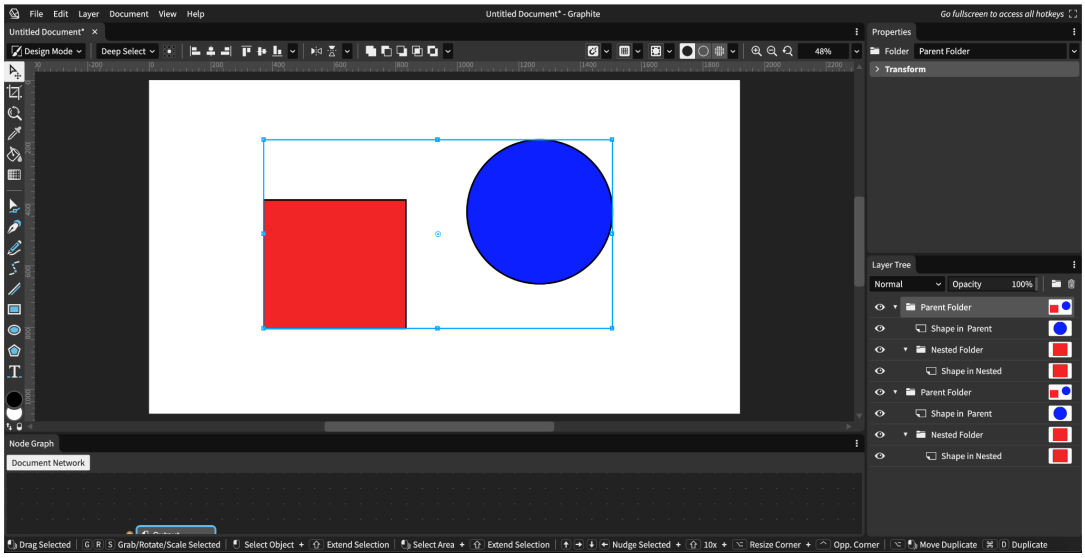
Duplicating a folder should also duplicate the children #1225:

https://github.com/GraphiteEditor/Graphite/issues/1225
Authors: Chris
Overview: When duplicating folders recursively collect all of the nested folders and layers.
Work: In the graphite layer system, we have two types of layers: a folder or a shape. Prior to implementing this feature, the duplication would have the same behavior for both shapes and folders. I worked on adding this distinction for folders, where duplicate folders would contain the

same shapes/folders that existed in the layer we duplicated. To do this, I had to create a recursive function that would collect all of the children layers by searching through the duplicated folder and all of its nested folders. This code has been merged into the master branch.



Before duplication



After duplication

Playable at: https://dev.graphite.rs/

Add line extension snapping mode

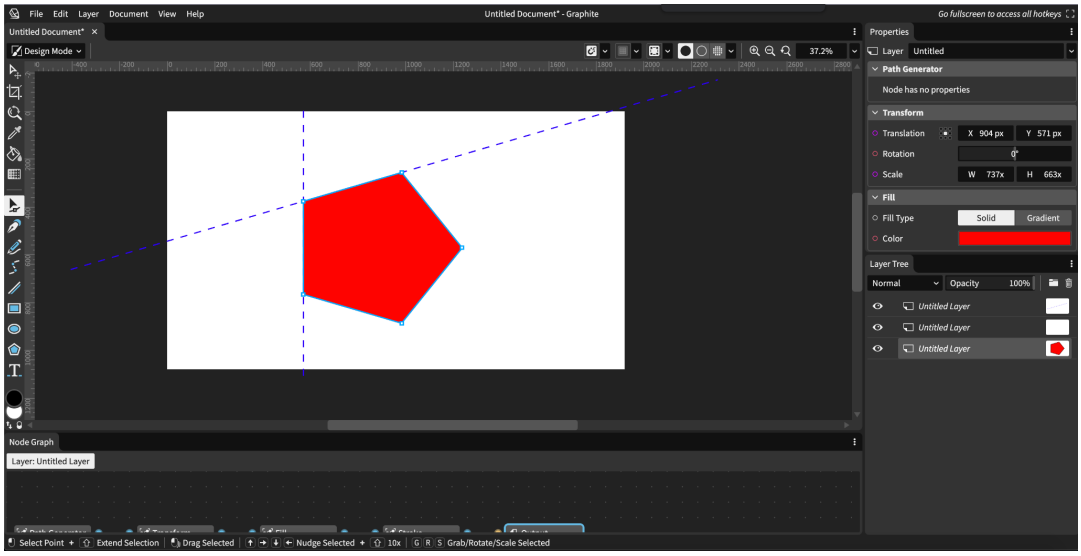
Pull Request: https://github.com/GraphiteEditor/Graphite/pull/1285

- Discussed issue during a meeting with Keavon

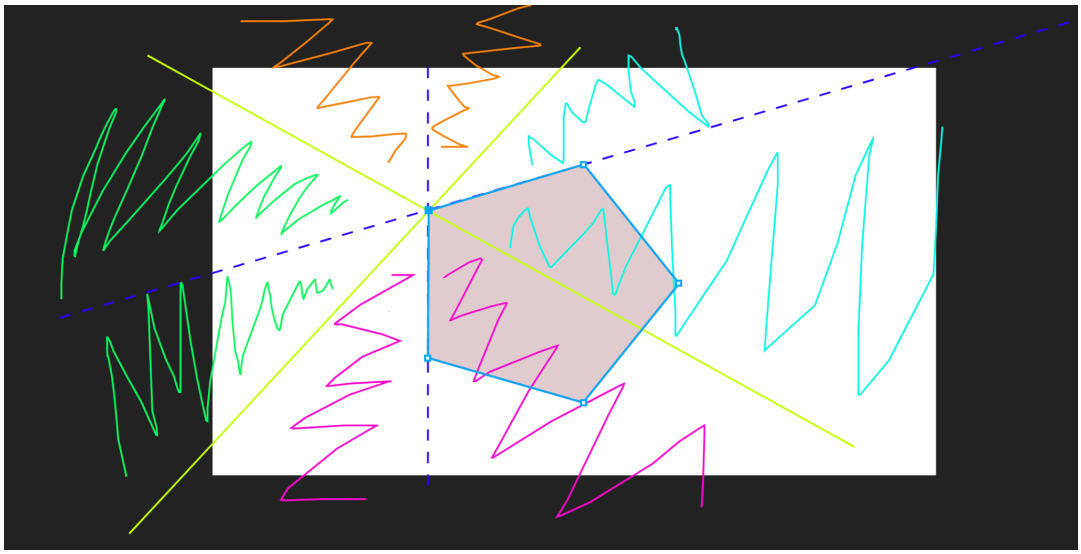
Authors: Chris

Overview: When dragging anchor points (vertices) with the V key pressed will enable snapping to whichever of the two imaginary lines the being-dragged point's current position is closer to.

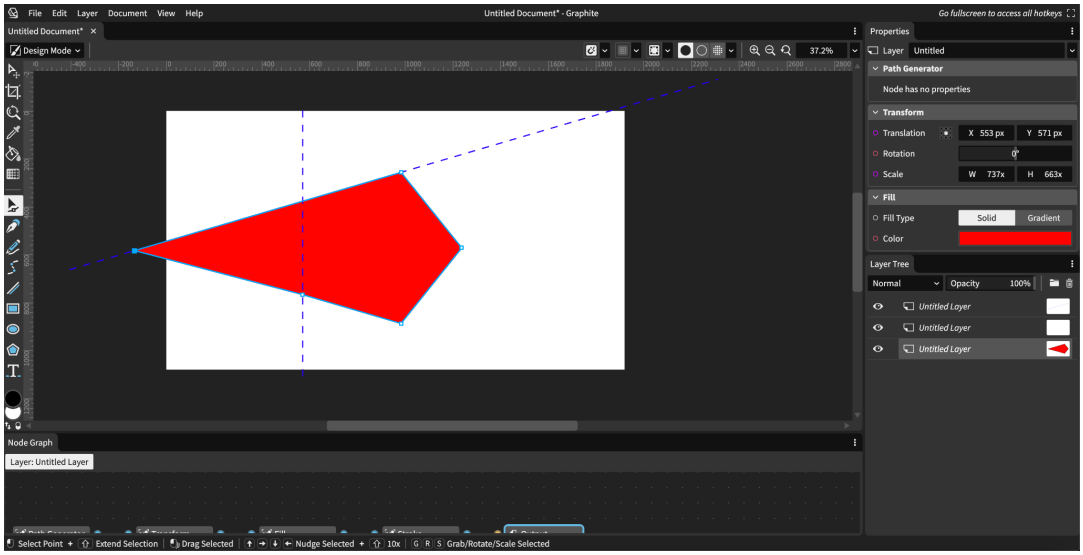
Work: My plan for this was to use the slope and the y-intercepts of the dragged anchor point with one or two of the nearby anchor points to create two imaginary infinite equations of lines representing the line extensions. To determine how the anchor points position moves, we use the cursor's position in combination with the infinite lines to project the anchor point's position onto the infinite line that our cursor is closer to. As a result, we can manipulate a shape's transform while maintaining the original angle. While dragging an anchor we can press and hold the key V to snap our anchor points to the infinite line. This code is being reviewed as part of our submission



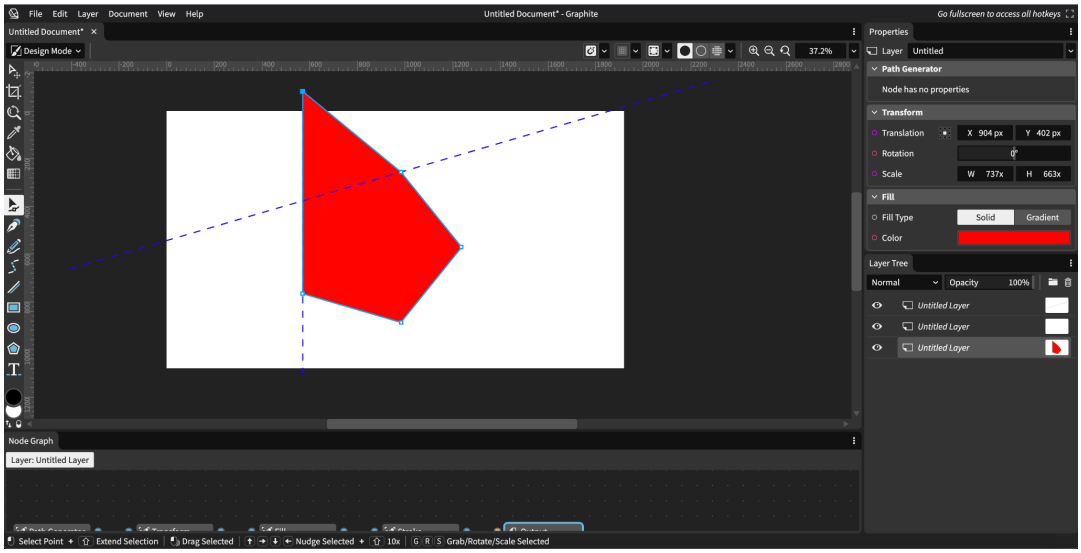
The blue dashed lines represent the infinite lines our anchor snaps to



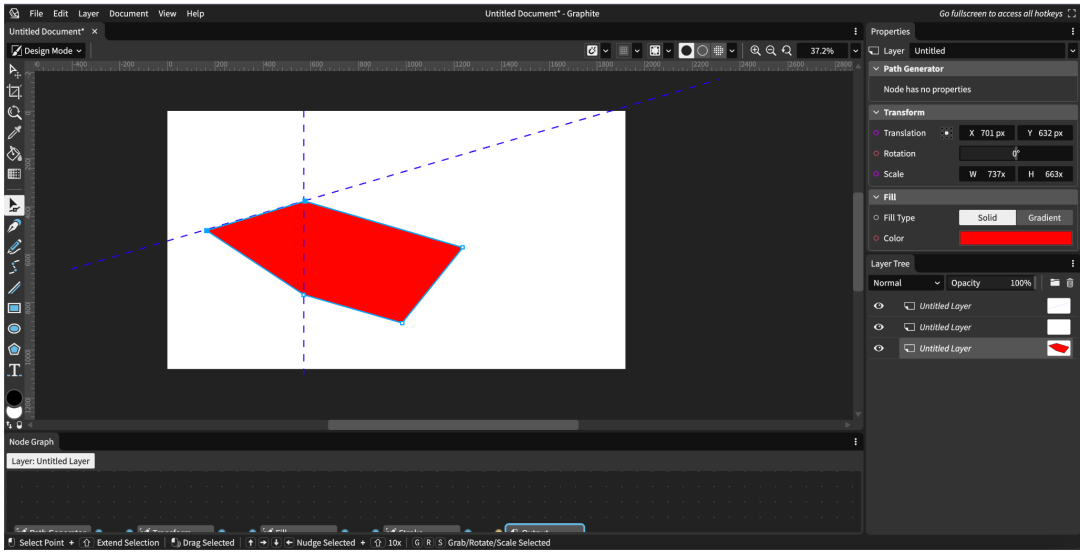
Our cursor's position determines which direction we extend (e.g. moving our cursor into the green will snap our anchor to the blue line left of the starting anchor position).



Moving our cursor to the left



Moving our cursor up

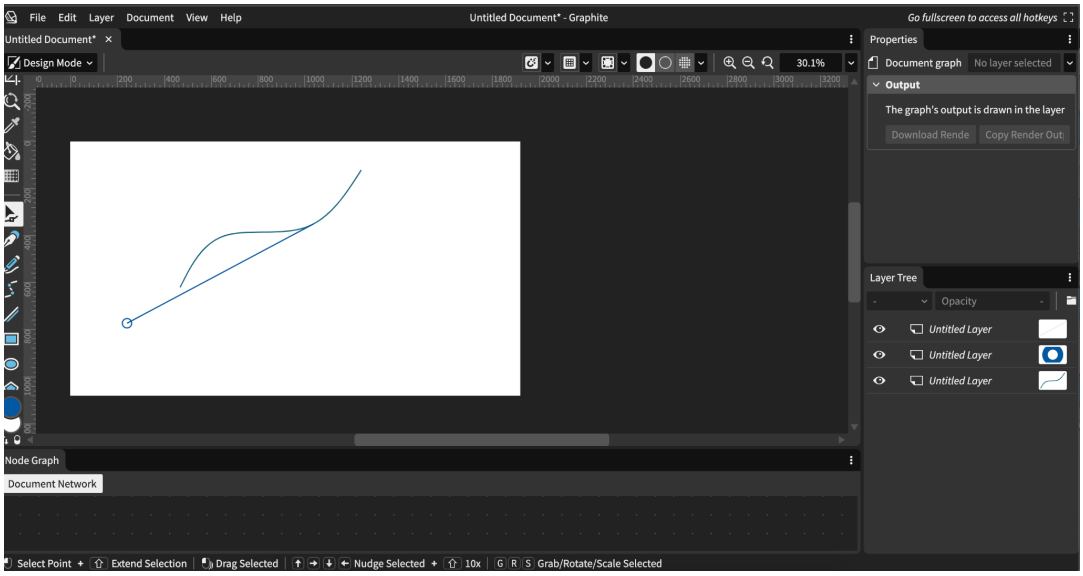


Line extending when multiple anchors selected

Playable at: https://82b24e51.graphite.pages.dev/

Tangent Snapping

Pull Request: https://github.com/GraphiteEditor/Graphite/pull/1290
Authors: Ollie
Overview: When creating/dragging a line, allow the user to snap to the point on a cubic bezier curve where a tangent would be created through the curve. This issue was very math based and took us a while to get the help we needed with the actual calculations, which involved solving quartic functions and using the Bezier curve equation to solve for points on the curve. Once we found the points on the Bezier curve where a tangent through our line would occur, we added visible candidate points to snap to.



This code is being reviewed as part of our submission
Playable at: https://4ac67332.graphite.pages.dev/

Verification

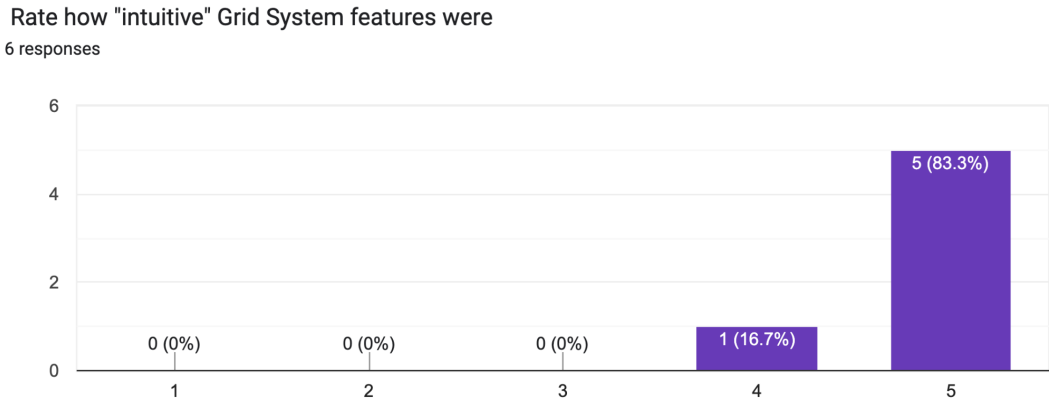
Testing:

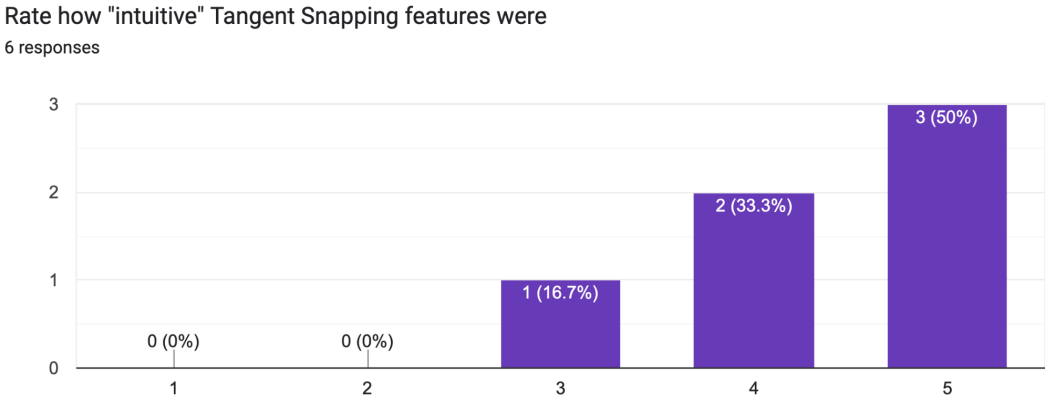
To verify the useability of our features we added, we conducted play tests to gather feedback and assess how user friendly our contributions are. Our users for our play tests are Cal Poly students with varying experience with Adobe Illustrator or any other similar graphics editor. Our goal was to make sure that the features we added were easy to learn and understand for the average user.

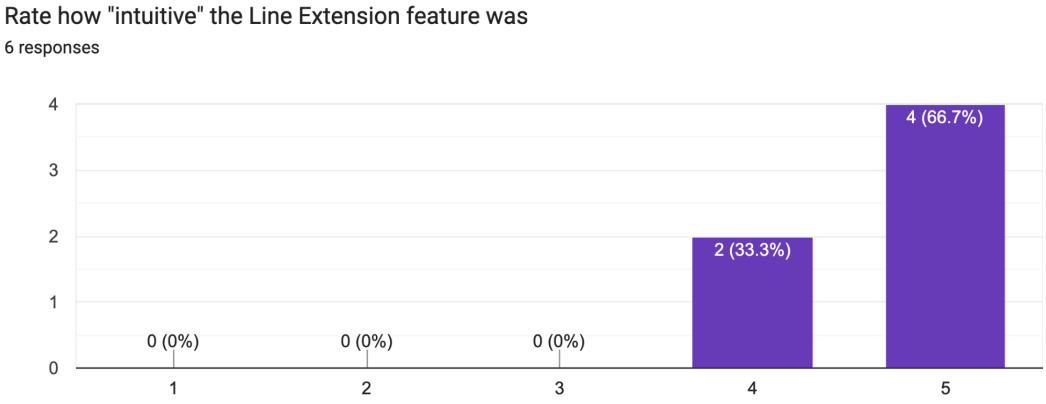
To gather feedback, we had each of our testers playtest each of the five substantial features. The features we tested were: grid system, tangent snapping, line extension snapping, grab/rotate/scale, and shallow select We narrowed our testing to these features because they had enough user interaction that could be tested for their usability. For each of the features we created scenarios for our testers to complete and provide feedback on how intuitive they were. For the grid system, line extension, and tangent snapping features we created icons using a set of tools and had our users replicate the icons. For shallow select, we had created a demo .graphite file designed to test the selecting of layers within folders. After

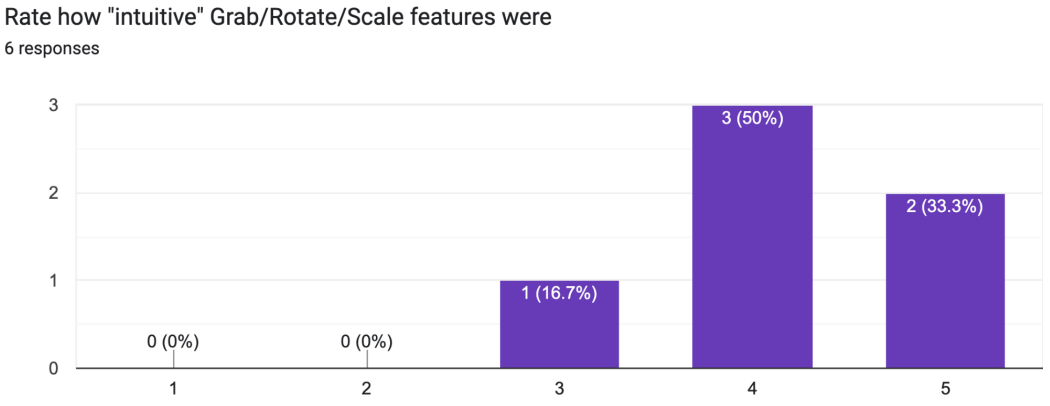
our users completed the scenarios, we had them fill out a google form. Here are the results from those questions:

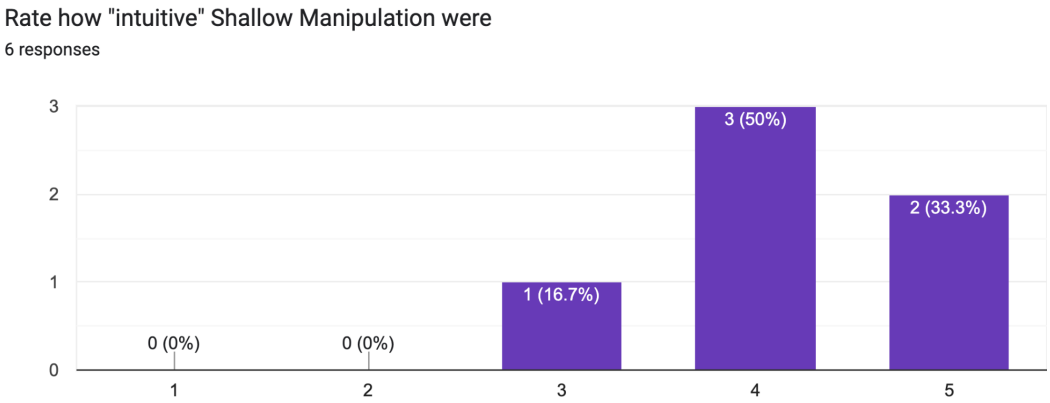
Results:

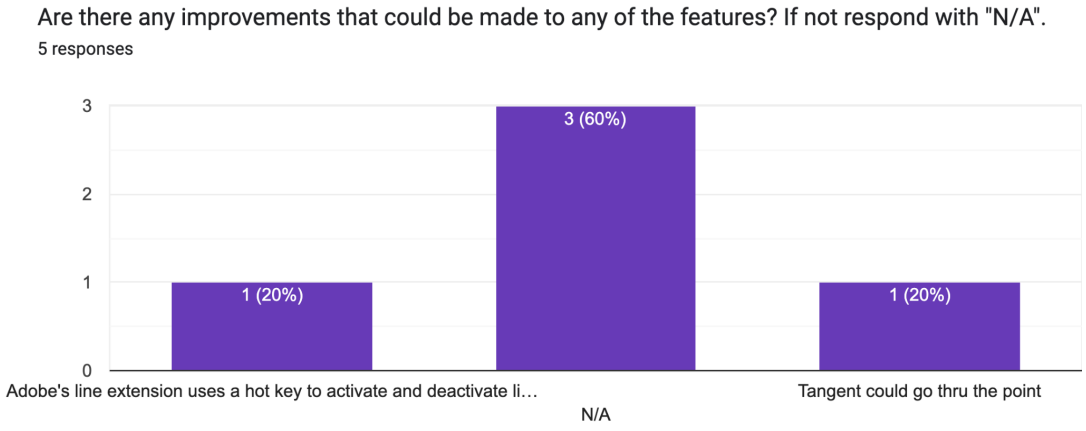










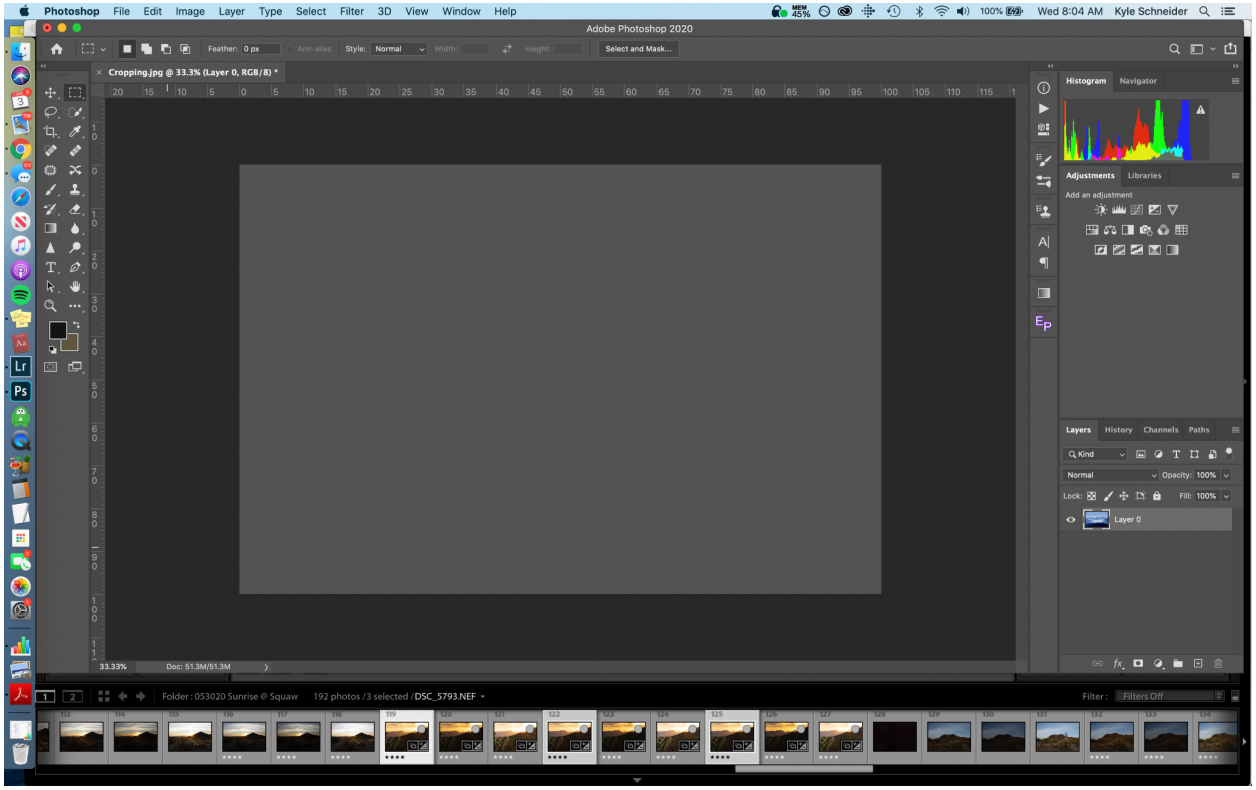


After having our work playtested and collecting feedback, we found that our users were able to successfully walk through the demo and complete the tasks we gave them. Despite the varying experience, our users found the five features to be intuitive and helpful. Our testers appreciated the

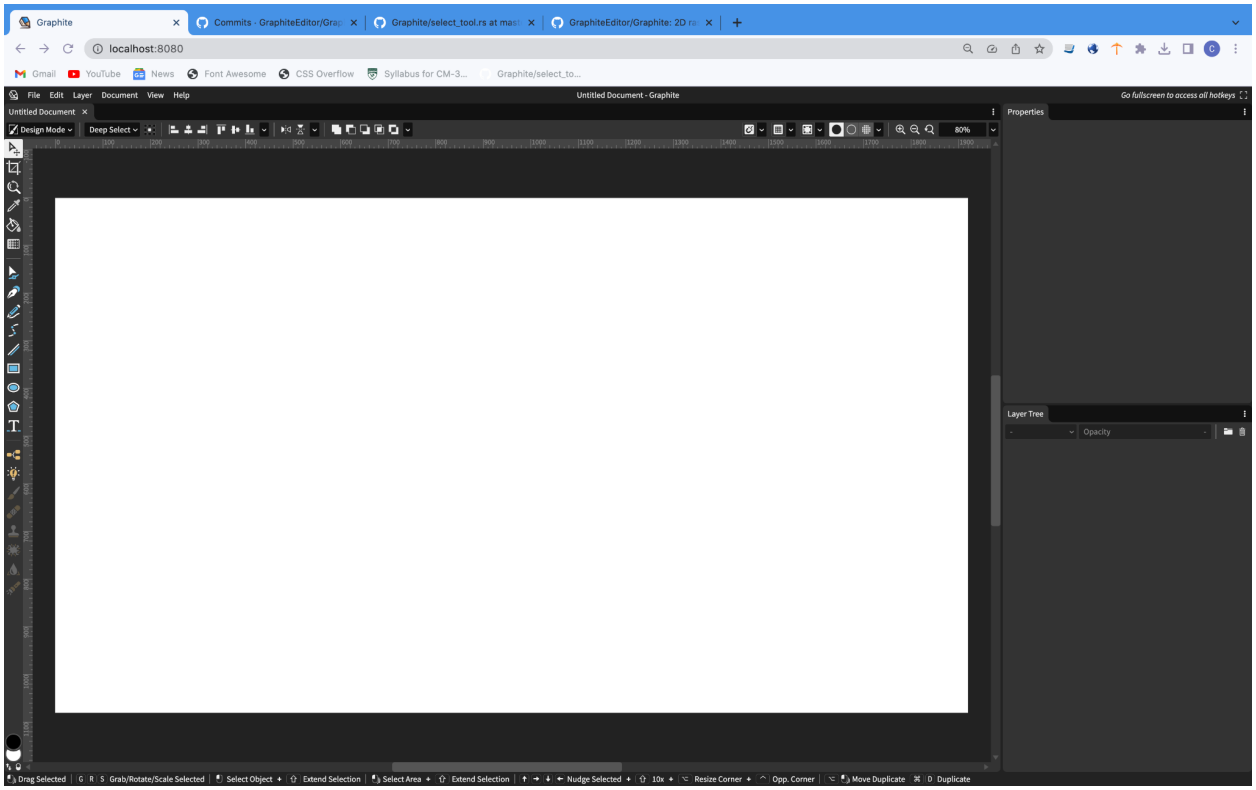
convenience that the tangent snapping and line extension snapping provided, removing the need for manual adjustments. As for the other features, our testers gave us positive feedback stating that the features seemed helpful and easy to understand. Overall, the feedback from the playtests indicated that our efforts in designing intuitive features were successful.

Analysis

Our goal for this project was to dive into the large Graphite codebase and get an understanding of the tool system, as a precursor to our overarching goal of creating a Snapping/Grid system for the Graphite editor as a whole. We succeeded in what we set out to do which was to complete a series of issues for the project that were related to various tools. We created solutions for issues such as grid/axis snapping, nudging points/layers, group vs individual selection of layers, rotating/grabbing/scaling points, and creating a snapping/grid system. The solutions we created were all scalable as they reused existing functionality from the codebase and largely our work was the product of using the resources from functions already created by previous contributors to the project. People find this product even more exciting and valuable than I realized when we signed on, everyday eager contributors join the Discord looking to help further Graphite just because they are interested in the modern design principles implemented. The user interface is very intuitive for anyone who has used an editor like an Adobe product as we used many of the same symbols and icons to prevent any confusion (Adobe Systems, n.d.).



Adobe Photoshop interface



Graphite Interface

Future Work

As most of our work this quarter was a precursor and foundation-setter for our main goal of creating a Snapping/Grid system we still have much to accomplish. But for this quarter we succeeded in solving all the issues and tasks we took on and feel confident our code can be used to further the codebase and lead to more innovation with this new editor.

Conclusion

This quarter we are proud to say we accomplished the tasks we set out to do. Each of us worked both together and individually at some point on the code and through our pair programming we were able to spot out errors much faster, the second pair of eyes is invaluable. The teamwork was especially important since we were all learning a new language with Rust and trying to familiarize ourselves with the extensive codebase which at first was a bit overwhelming.

Bibliography

Adobe Systems. (n.d.). Adobe Illustrator (Version 27.6) [Software]. Retrieved March 24, 2023, from https://www.adobe.com/products/illustrator.html

SideFX. (n.d.). Houdini (Version 19.5.640) [Software]. Retrieved March 24, 2023, from https://www.sidefx.com/products/houdini/

The Foundry Visionmongers Ltd. (n.d.). Nuke (Version 13.0) [Software]. Retrieved March 24, 2023, from https://www.foundry.com/products/nuke

The rust programming language. The Rust Programming Language - The Rust Programming Language. (n.d.). Retrieved March 24, 2023, from https://doc.rust-lang.org/book/