3Dev:

3D Graphical Development Software

Jamie Andrews

Analysis

Problem Identification

Background

Computer generated images (CGI) are used across a number of industries such as film production, architecture and game development. In previous years, creating engaging and clear visuals was a task that had to be done manually - whether that be an architect sketching every last detail of a building or an animator drawing every single frame of an animation.

This greatly limited these industries for years, as vast amounts of time and money were spent on human labour in order to complete these tasks. This began to change with the introduction of computer graphics. From the late 1960s, methods such as ray-casting (1968) and z-buffering (1974) were developed to render 3D scenes, but these methods have their limitations in the complexity of the scenes they are able to generate. Early ray-casting only considered primary rays (rays from the camera) and not secondary rays such as reflections or shadows.

These features are essential for generating realistic scenes which closely resemble true textures and lighting, making it a viable technology for use in more professional industries.

More modern approaches such as ray tracing allow for these more complex effects and can also be optimised to render more detailed scenes without the need for such ridiculous processing times. However, many modern applications of ray tracing require expensive hardware which is not accessible to those who want to explore 3D visual industries in a recreational or semi professional capacity.

Therefore, I aim to create a system which can generate and render complex 3D scenes on modest modern hardware, using current ray tracing techniques. I will be focusing on optimizations which provide the best trade off between reduction in run time and simplicity of code development. This is because I plan on making the project open source in order to maximise accessibility for those with differing use requirements - the more straightforward the code, the easier it is for someone to specialise it for their particular use case.

Computational Methods

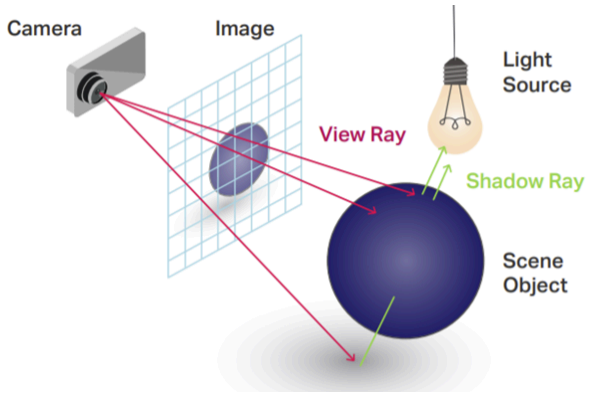
Ray tracing works by calculating the path of a light ray from the camera centre (eye point) to objects in a scene, through every pixel in the frame being rendered. This means performing lengthy vector calculations to find what objects each ray intercepts with and what colour value that ray represents for every single pixel. To do this without the use of a computer would take countless man hours in order to render just a simple, low resolution image.

Each ray can be thought of as a function, $P(t) = A + bt$, where P represents a 3D position along a line, A is the origin of the ray, and b is the direction of the ray. t represents a point in time, different values of t correspond to different positions along the path of the ray. Rays will be represented in code using a ray class, where A and b are vector objects, defined by its own class, and attributes of the ray class.

We now have to do 2 things to render an image:

- ➔ Calculate the ray through the pixel, from the eye point
- ➔ Compute the intersection points of the ray with objects in the scene

For this I will need to setup a camera (located at the origin) and viewport (through which the rays will pass into the scene)



One of the primary advantages to using a computer for this kind of task (aside from the fact that each individual calculation can be performed orders of magnitude times faster than the attempt of any human) is that, using modern hardware such as a GPU, methods such as parallel processing can be utilised to process many rays in the scene simultaneously. This is because the processing for the image can be broken down into a large number of rays, which can be calculated completely independently of each other - since they are not dependent on the behaviour of other rays. Additionally, since the same kind of calculations are being run for every ray, the Single Instruction Multiple Data (SIMD) architecture of modern GPUs lends itself to the simultaneous calculation of rays at a much faster rate than that of a CPU, which is forced to perform calculations for each pixel much less concurrently due to its limited number of cores.

When calculating the intersections between rays and objects, one option is to just have an array full of objects in the scene, all with their own different attributes. It is preferable to abstract all of the objects into a parent class which represents anything that can be hit by a ray. This means that later on in development it becomes easier to implement functionality around other types of hittable entities such as volumes. Additionally, we can have an abstract material class with attributes to describe how a ray is scattered once hit, which encapsulates the unique behaviour of different material types. This is opposed to having a universal material type with many different attributes that can be acknowledged or ignored for materials which have different properties.

In order to create smoother transitions between objects in the foreground and background without dramatically increasing the resolution, I can make use of antialiasing which takes multiple samples for each pixel and calculates an average for the colour. This is ideal for a computational approach as it involves the integration of a continuous function of light which falls onto a discrete portion of the image - something that cannot realistically be done by

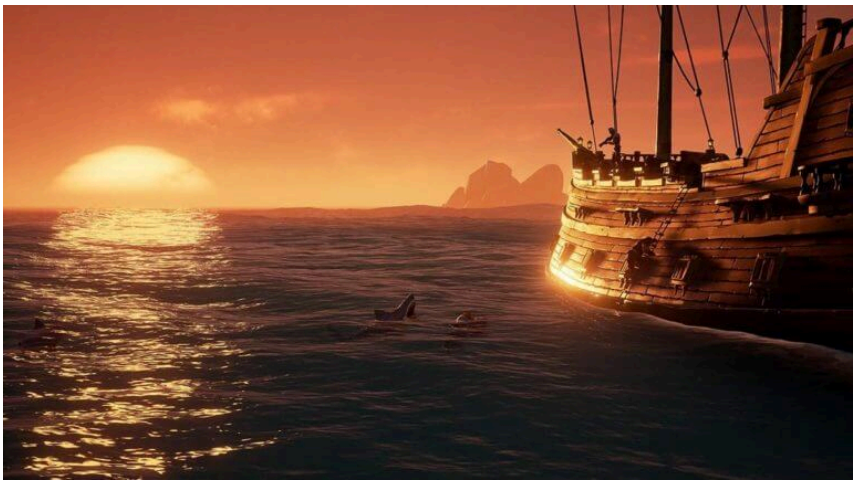
humans for detailed shapes. The most straightforward approach to this is to sample 4 points within the region bound halfway between the pixel and its 4 neighbours in each direction.

Overall, my project is amenable to computation methods because it is contingent on the processing of copious amounts of the same type of complex calculation. This is something the human brain is ill equipped for as we are not adept at thinking concurrently when performing long, complicated tasks such as vector calculations - unlike modern computing hardware that can handle much larger sets of tasks at a consistent rate.

Stakeholders

Stakeholder 1

My first primary stakeholder for this project is **Martin Linklater**, a game developer with over a decade of experience on mainstream 3D games such as Sea Of Thieves.



Martin needs something which can render 3D scenes, involving a range of features such as:

- ➔ Varieties of textures
- ➔ Volumes (such as smoke, fog or clouds)
- ➔ Varieties of shapes
- ➔ Lights
- ➔ Support for large quantities of objects

He plans to combine my scene renderer with animation software to create semi-realistic, cinematic cutscenes for his 3D indie game. This system would be suitable for his use case as he will need to make use of more realistic rendering effects on his personal computer.

My software should be a useful tool in the creation of his game and should not be the reason for any bottlenecks in the speed of his development. This means my ray tracer must be able to process and render a scene in reasonable working time so that Martin can swiftly move onto tasks which depend on it.

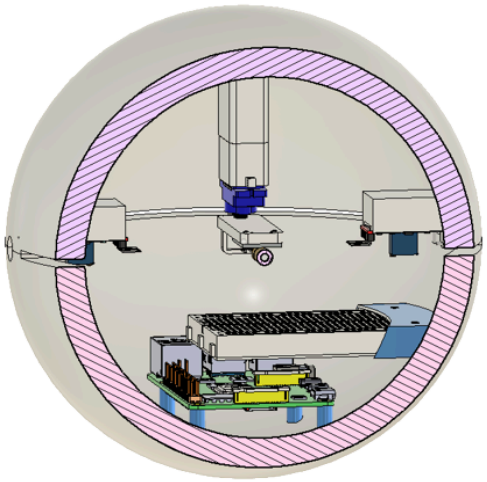
Martin will need to construct a variety of different scenes for different points throughout his game. This means one of the essential features for this project is an intuitive interface in order to create, save, and customise different scenes. Martin will then be able to easily keep track of his different cutscenes and edit them where necessary as the development of his game progresses.

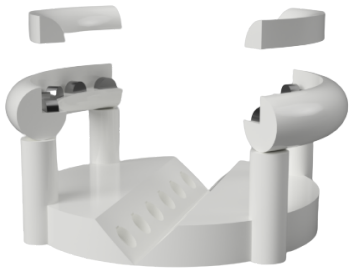
Martin is a developer who works on large professional games but also enjoys working on smaller scale personal games. My project will attempt to elevate the quality of his personal game graphics closer to that of his larger scale endeavours.

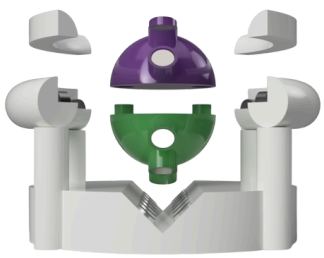
Throughout the development of my project Martin will be interviewed 2 times - once during the research stage of my project (To clearly outline what features he needs for the software to meet his needs), and once at the end of my project during the evaluation stage (After he has completed the final black box testing of my finished product). Martin will also be doing user acceptance testing after the rendering sprint of my project to provide feedback on the quality of the ray tracing and the sufficiency of the features offered by my renderer to meet his needs.

Stakeholder 2

My second stakeholder is **Jason Wang** who is an A-Level student with a passion for electronics projects (which he models and 3D prints plastic parts for) and making organic 3D models.







Jason needs a program which he can use to combine shapes to make basic prototypes for his electronic project models. He will then render these to get a preview of what the part will look like before modelling it fully in Fusion360. Additionally, Jason needs to go through a similar process for his organic model projects to get an idea for how complicated the shapes and parts of his model are likely to be, in order to decide whether or not the project is feasible.

Jason needs to be able to render a variety of materials which he might decide to use to build his components out of. He would also like to see the prototypes for his organic models rendered in as much detail as possible as he has invested in a high quality monitor for these projects. This means that 4k rendering support would be necessary to fully cater to his use case.

My project is very suitable for Jason's needs as I will be focusing on the ray traced renders as the primary feature of my project with a simple scene creator tool to make simple composite shapes and scenes.

Jason is also going to be interviewed during the research stage of my project, in order to identify the needs for his specific use case, and during the evaluation stage, to give a review on how well the software served his requirements. Additionally, Jason will be doing user acceptance testing like Martin but after the scene creation sprint so that he can give feedback on how successfully he was able to put together objects and scenes for his prototypes.

General Stakeholders

Due to the computationally intensive nature of ray tracing and its prominent use of parallel processing, it is necessary for the user to have a dedicated graphics card installed into their computer. However, due to the graphical nature of many industries which my software will apply to, this is a criterion that the majority of my prospective clients will already satisfy.

The target for my software are independent workers/freelancers as well as people who are passionate about exploring 3D graphics. Martin & Jason are going to represent people from my target audience in order to give insight into how my system is used by someone developing 3D graphics for an individual project.

However, I will also be providing my software to a variety of individuals - such as friends, family and other students - in order to perform alpha testing after the final user interface (& version control) sprint of my project. This allows me to gain a wider range of feedback on the work done in the other two sprints, as well as an idea of how intuitive my user interface is for beginner users who don't necessarily have much experience with other 3D graphics softwares.

Research

Existing Tools

Professional level software used for 3D image rendering includes tools such as **Blender[1]**, **D5-Render[2]** & **Fusion 360[3]** which all have slightly different advantages in different areas - making them more suited to different industries.

Scene Analysis

D5-Render is predominantly used for real world modelling such as architecture and open world landscapes or natural environments. I will review examples for both of these as well as some additional features which the platform offers.

D5-Render example scenes

There are 4 main features of this render that have a huge impact on the image:

- ➔ **Water Reflection**

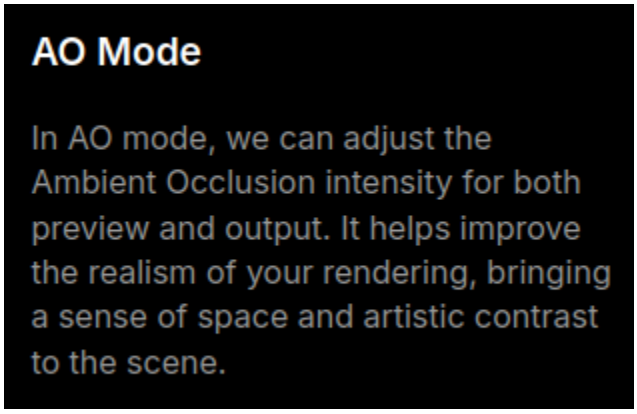
The reflections of the forest in the stream are darker than the environment itself which helps make the water easily distinguishable from the objects around it. This makes the image look more realistic because in real life, some of the light is refracted through the water rather than all of it bouncing right off the surface. Additionally, there is a good balance between the clarity and blur of the reflection - since the surface of water in real life has ripples and imperfections that distort the reflection, while still being more clearly reflective than other non-matte surfaces such as metal.

- ➔ **Direct and indirect lighting**

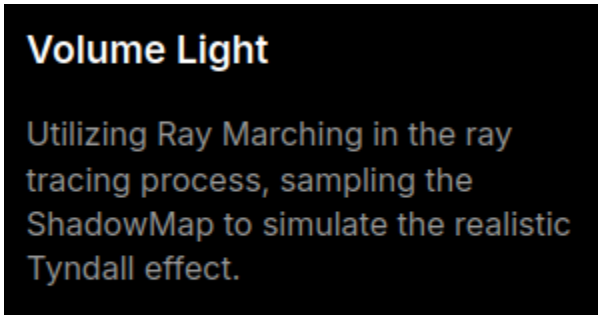
This scene makes extensive use of shadows and rays of sunlight through the trees to highlight key areas of the image such as the animal drinking from the lake. However, there is also use of the indirect blue light coming from the rock, which does not heavily affect much

of the scene around it other than the water but does contribute to the overall feeling of the environment.

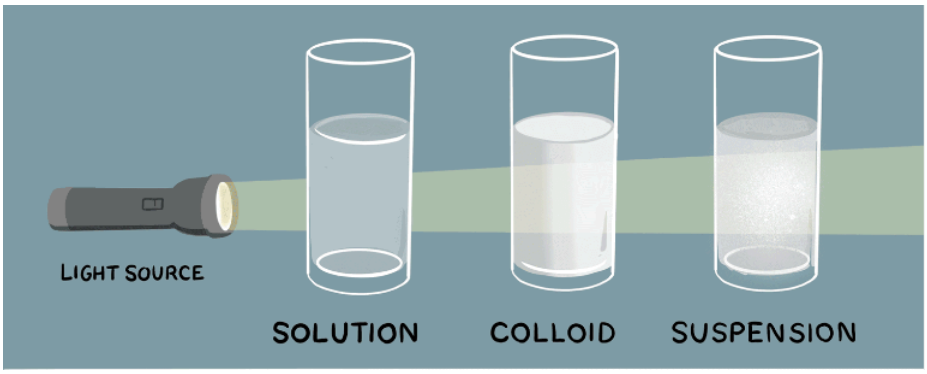
D5 also offers other lighting effects, as described on their feature list[4] as follows:



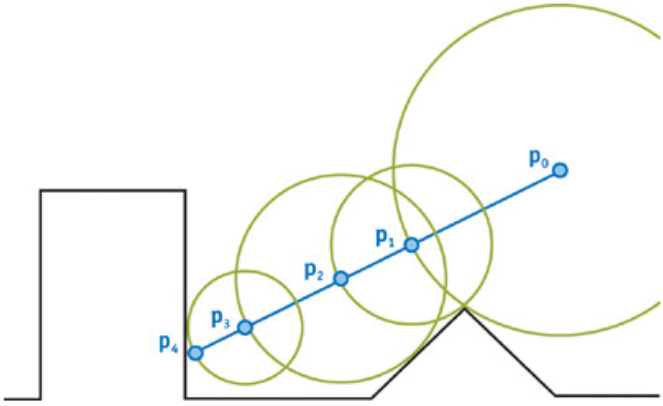
Ambient occlusion involves calculating how exposed each point on the surface of an object is to ambient light present throughout the scene in order to make shadows appear more smooth and realistic.



The **Tyndall effect**[5] is the scattering of light from fine particles suspended in a fluid which makes the light more visible as it passes through.



D5 achieves this by making use of **ray marching**[6] which is a rendering technique that uses distance functions between a point and objects in a scene to determine how far a ray can travel without hitting an object. This is done by monitoring the minimum distance between the leading point on the ray with all the objects in the scene until it reaches a sufficiently small value.



From GPU Gems 2: Chapter 8.

➔ **Fog (Volumes)**

There is an increasing gradient in the thickness of fog noticed in the image with depth, this draws more attention to the foreground and the key features of the scene.

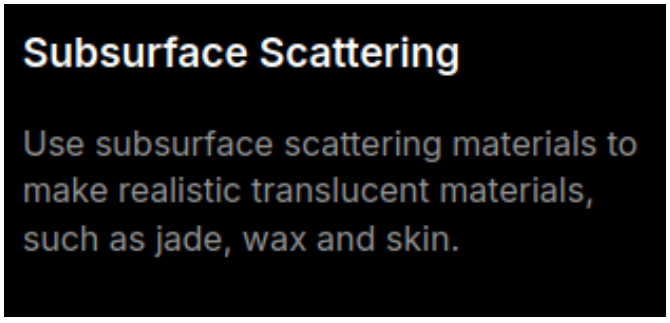
➔ **Wide range of models**

As well as all the effects and rendering techniques, a crucial factor which made this scene possible is the wide array of 3D models the designer was able to incorporate into the scene in order to give it character and make it more interesting to keep looking at for longer. It is important that users of my software are not bottlenecked in the uniqueness of their scenes by the limited number of shapes/models available. Since D5 is not primarily a modelling software (unlike the other two competitors on the list), it has large amounts of preset models available as well as the ability to import your own 3D model files into the program. I believe this importing of models will be most beneficial to my software as it prevents a bias towards one particular industry due to unbalanced support for different kinds of scenes which use different models.

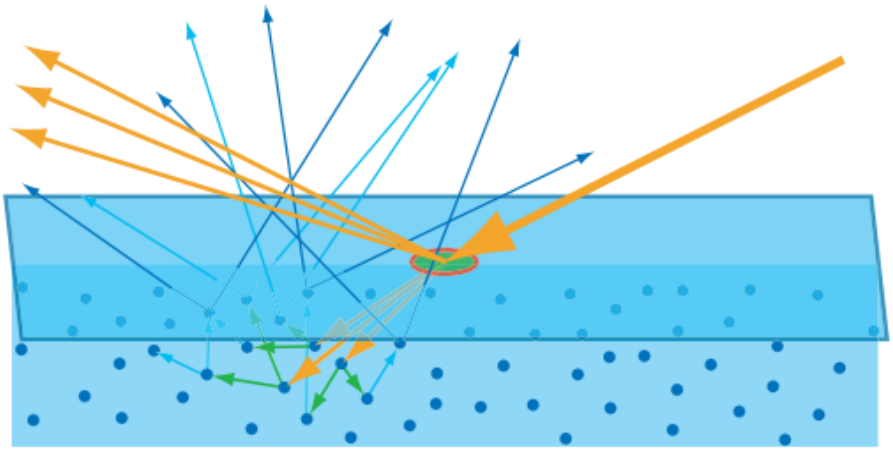
Similarly to the previous one, this image makes use of harsh lightning and shadows to create the desired aesthetic tone for the scene. This is something that I wish to recreate in my software by allowing the user to change the intensity and colour of light sources to fit the theme of the scene best. This scene also puts a large emphasis on textures, as both the buildings and general infrastructure visible are composed of a set of different materials. Because my user base is fairly open towards people from different industries, I think it would be equally useful to incorporate **importable textures** (as well as implementing a **texture mapping** system) in the same way I will include imports for 3D models.

Feature Review

D5 Also includes a range of other features which I plan to fully or partially recreate:



Subsurface scattering[7] would be useful in my software to render more realistic looking skin as some users (such as my primary stakeholder Martin) will want to render characters in high definition cinematic scenes. This technique works to render translucent materials by reflecting some rays off the surface of the material and diffusely scattering the rest after they pass through a portion of the object.



Cinematic Camera Effect

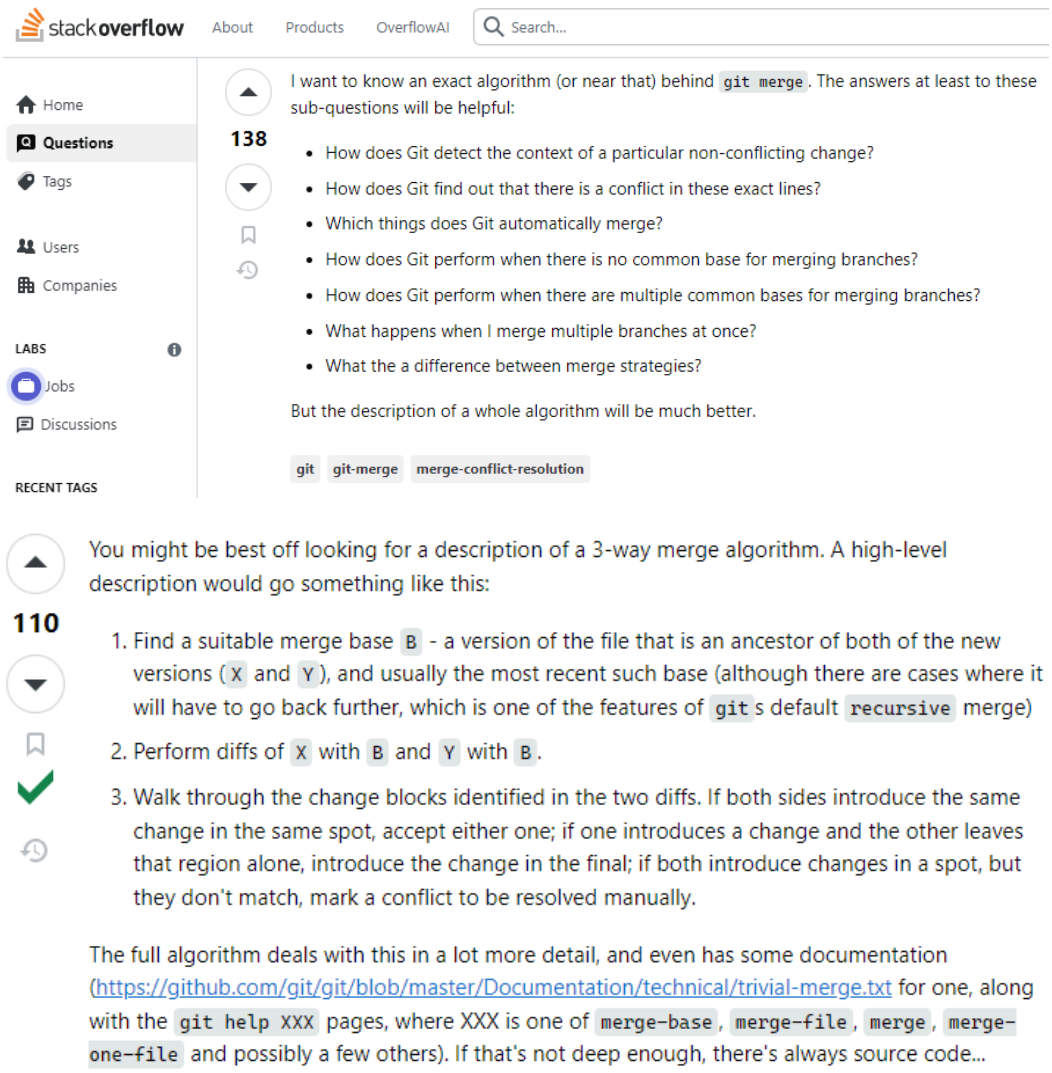
Create a cinematic scene through parameters including Bloom, Lens Flare, Chromatic Aberration, and Vignette.

Cinematic camera effects such as lens flare may be beneficial for some cases but are often utilised more in video rendering as it can draw too much attention away from the focus of the image unless only visible for a short period of time.

Merge Projects*

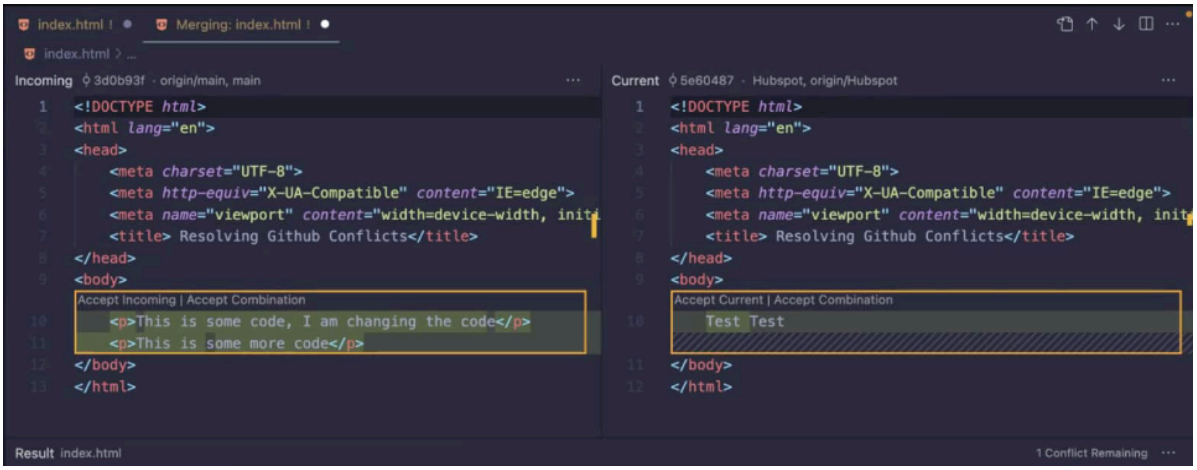
Teamwork made easy. Support merging up to 10 projects at the same time.

There may be teams of developers who are working on different aspects of a complicated scene and want to work independently of separate sections before collecting their work into one scene. **Merging multiple projects** across different accounts (or the same account) allows users to do this without having to use other third party software.



This merging feature is reminiscent of the merge feature for github repositories so I explored a stack overflow forum post to identify the main steps which a merging algorithm would need to carry out in order to produce a functional file which combines the two projects.

For any objects which overlap when merged, I plan to recreate the github merge conflict flag by highlighting conflicting objects in a visible bright red colour and introducing a conflict tab with a list of all the conflicting object sets for the user to go through and select the object(s) they wish to keep and objects they wish to discard.

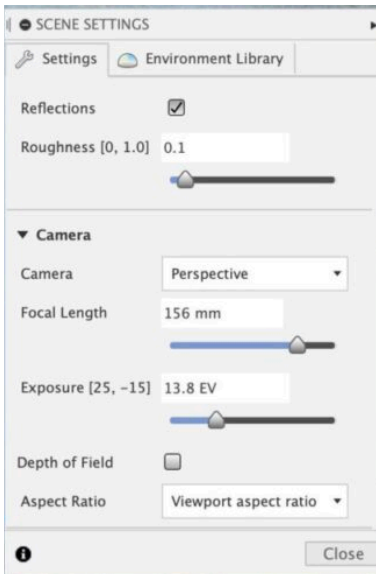


Stakeholder Interviews

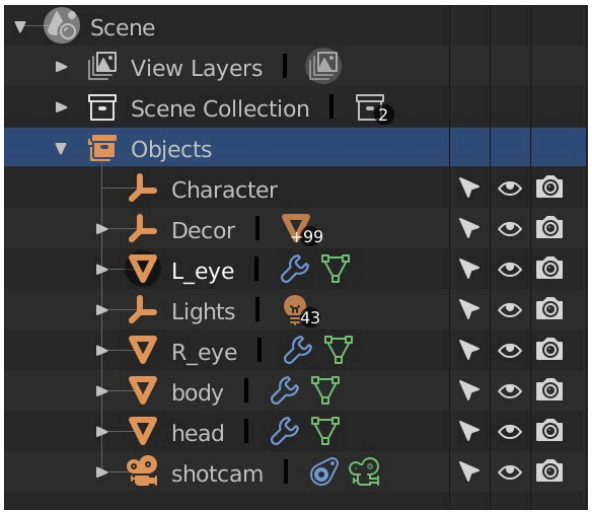
General Experience

➔ **What specific features do you find most useful in Blender and Fusion 360?**

- *"Render sliders (nearly all of them have it) just allow you to select for what refinement of renders you want, and having a slide made it very easy."* - Jason



- *"I find the object management tab in blender very useful to organise everything I have in my scene - particularly small objects which can be easy to lose track of amongst the rest of the objects"* - Martin



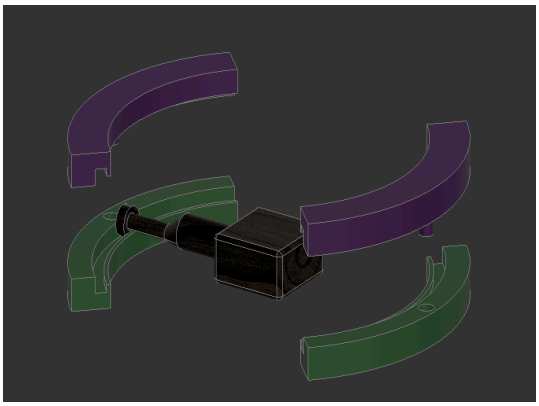
➔ **Are there any features or tools you wish these software packages had?**

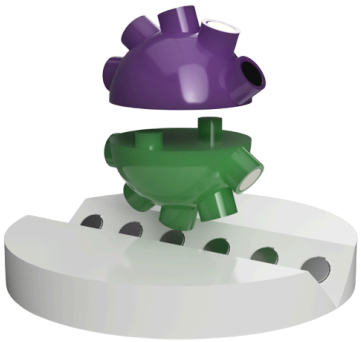
- *"Blender covered everything i needed for organic models and Fusion 360 covered everything for solid or mechanical things"* - Jason
- "" - Martin

Rendering Needs

➔ **What types of scenes or objects do you typically render?**

- *"Still plastic parts I 3D print for electronics projects or organic models like animals just for fun"* - Jason





- *"Mostly putting together open scenes with a few characters and props"* - Martin

➔ **What are your primary goals for rendering quality and speed?**

- *"I don't have many time restraints, I am more interested in the render being as high quality as it can be"* - Jason
- *"Time isn't an issue for still images its just quality I care about"* - Martin

I will prioritise generating the highest quality renders possible over the time it takes to render the final image (within reason).

Workflow Integration

➔ **What types of import/export capabilities are essential for your work?**

- *"I need to be able to import custom textures and I only really need to be able to save the final render as an image. It would be useful if I could choose the aspect ratio of the picture though."* - Jason

  I will create a dropdown when rendering an image to decide the aspect ratio of the image - changing the aspect ratio will have no impact on the resolution of the render so that all options will produce the same image quality.

- *"Importing all the basic stuff like textures and backgrounds is all I need, as well as saving with all the usual image file types like jpg and png"* - Martin

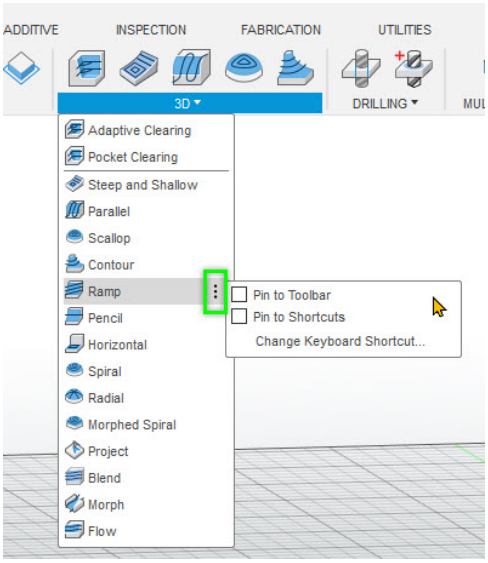
  I will implement an import option for backgrounds and textures which will add a file from the users computer to the appropriate database. These imported textures/backgrounds can then be used for mapping in project files (as long as their files are the correct format).

User Interface and Usability

➔ **How important is the user interface to your workflow, and what aspects do you find most crucial?**

- *"These softwares often have a lot of features that can be hard to keep track of so I really like when it is very clear how to navigate to all of the key features"* - Jason
- *"In some blender dropdowns with lots of options, everything is crammed together with small writing which can be hard to navigate"* - Martin

  I will try to follow Fusion 360's dropdown menu style with spaced options and shorter naming styles. Fusion 360 also implements extensive use of small diagrams in order to illustrate the function of different buttons and options, this is something I may be able to take advantage of depending on the complexity of implementation.



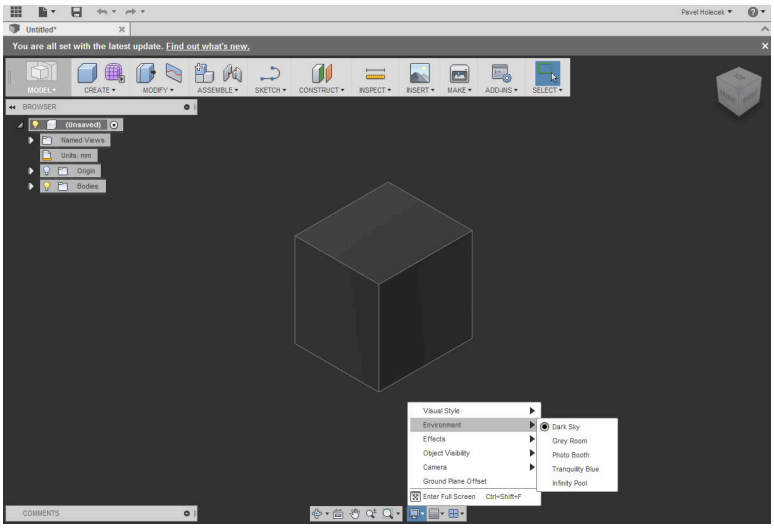
➔ **Are there any UI features in Blender or Fusion 360 that you find particularly effective or frustrating?**

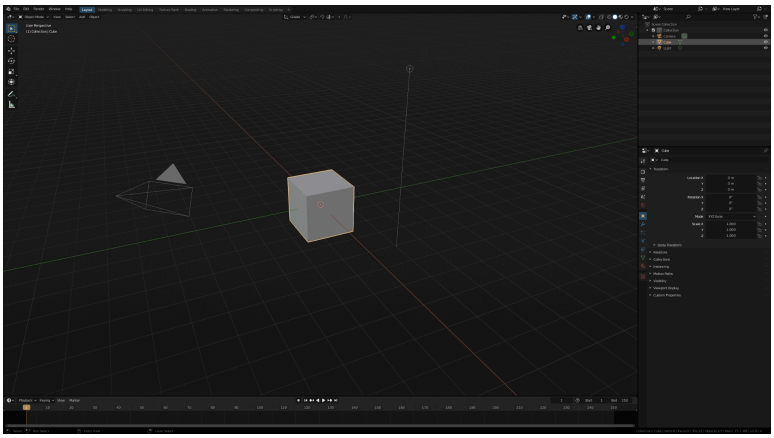
- *"There is a large database of textures which I find useful but how it's arranged isn't the easiest as it feels like everything is everywhere. The environment was also pretty cool but had the same issue with user experience"* - Jason

  Blender displays an extremely extensive database of textures in large lists that are often difficult to navigate through in order to find the texture you are looking for - especially if you don't know what it is named as. To combat this I will attempt to implement a searching feature to the textures database which allocates multiple descriptors to each texture which will be used for search referencing so that even if the user does not know the name of a texture they will still be able to search for it more easily.

- *"I work pretty late at night so having a dark mode is something I find pretty nice to have, the Fusion 360 dark mode isn't ideal though because only the environment is dark and not the tabs"* - Martin

I will try to adopt a more blender-style dark mode rather than the default fusion 360 dark mode which does not affect the entire interface





Collaboration and Sharing

- ➔ **Do you often collaborate with others on projects? If so, what collaboration tools or features are important?**
  - *“I don't really most of my projects are individual”* - Jason
  - *“Yeah I work with people on my team pretty often on developing characters”* - Martin

    I will allow multiple users to sign in on the same install of the program in order to work on separate projects. These users can then merge their project files, as mentioned under the “Existing Tools” section of my research.

- ➔ **How do you handle version control or scene management for complex projects?**

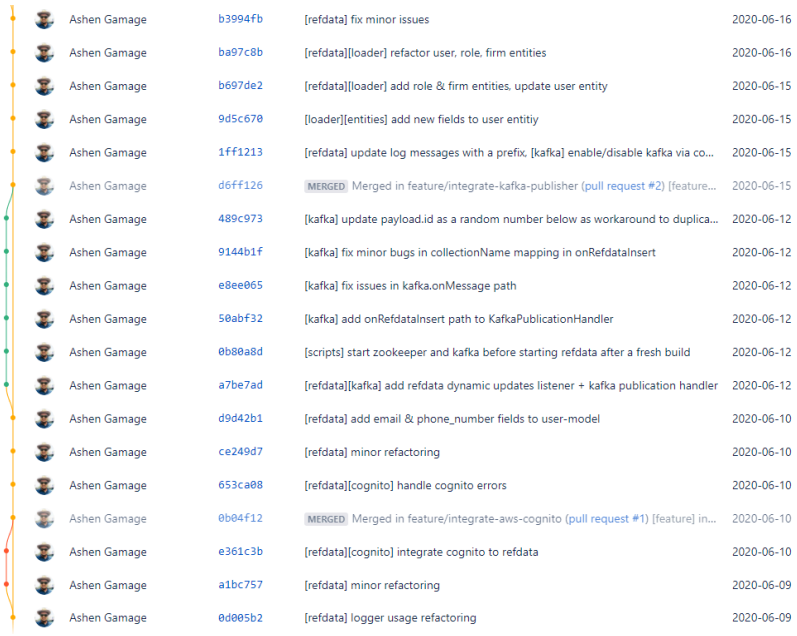
- *“I like being able to choose between saving projects as versions or overwriting the file, that way I don't lose old version of bigger more complicated projects but I also don't end up with lots of unnecessary files for a simple scene”* - Jason

  I will give the user an option when saving project files if they wish to overwrite the current file or create a new file with the updated content in the same folder as the current file

- *“We use GitHub to manage the versions of our model files and which versions of the game code they were developed around”* - Martin

  It may be beneficial to allow the user to save comments attached to each version of a project as they save it - similar to the Git commit message system. Following inspiration from Git, saving the date on which the version was saved could also be useful for 2 reasons:

  - ➔ Allowing the user to see the dates on which they saved a version lets them get a better idea of how on track they are with the progress they want to make on a project within certain periods of time.
  - ➔ Automatically displaying the version options of a project in reverse chronological order when the user wants to open one.



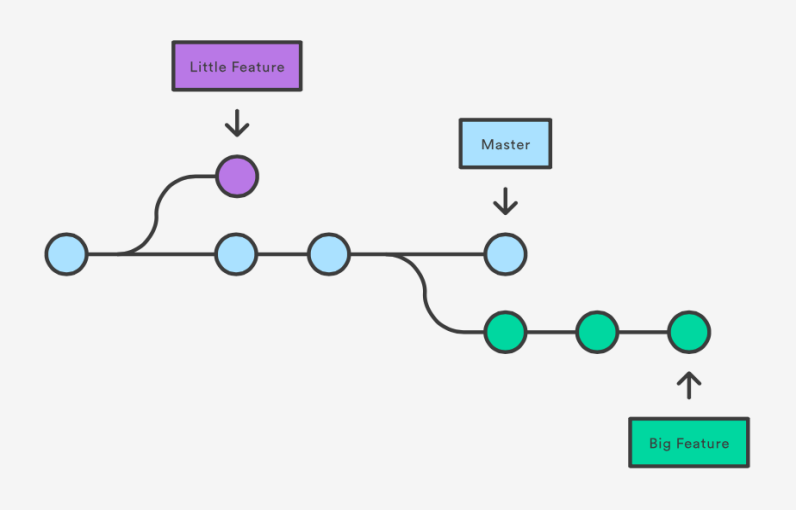
Ashen Gamage	b3994fb	[refdata] fix minor issues	2020-06-16
Ashen Gamage	ba97c8b	[refdata][loader] refactor user, role, firm entities	2020-06-16
Ashen Gamage	b697de2	[refdata][loader] add role & firm entities, update user entity	2020-06-15
Ashen Gamage	9d5c670	[loader][entities] add new fields to user entitiy	2020-06-15
Ashen Gamage	1ff1213	[refdata] update log messages with a prefix, [kafka] enable/disable kafka via co...	2020-06-15
Ashen Gamage	d6ff126	MERGED Merged in feature/integrate-kafka-publisher (pull request #2) [feature...	2020-06-15
Ashen Gamage	489c973	[kafka] update payload.id as a random number below as workaround to duplica...	2020-06-12
Ashen Gamage	9144b1f	[kafka] fix minor bugs in collectionName mapping in onRefdataInsert	2020-06-12
Ashen Gamage	e8ee065	[kafka] fix issues in kafka.onMessage path	2020-06-12
Ashen Gamage	50abf32	[kafka] add onRefdataInsert path to KafkaPublicationHandler	2020-06-12
Ashen Gamage	0b80a8d	[scripts] start zookeeper and kafka before starting refdata after a fresh build	2020-06-12
Ashen Gamage	a7be7ad	[refdata][kafka] add refdata dynamic updates listener + kafka publication handler	2020-06-12
Ashen Gamage	d9d42b1	[refdata] add email & phone_number fields to user-model	2020-06-10
Ashen Gamage	ce249d7	[refdata] minor refactoring	2020-06-10
Ashen Gamage	653ca08	[refdata][cognito] handle cognito errors	2020-06-10
Ashen Gamage	0b04f12	MERGED Merged in feature/integrate-aws-cognito (pull request #1) [feature] in...	2020-06-10
Ashen Gamage	e361c3b	[refdata][cognito] integrate cognito to refdata	2020-06-10
Ashen Gamage	a1bc757	[refdata] minor refactoring	2020-06-09
Ashen Gamage	0d005b2	[refdata] logger usage refactoring	2020-06-09

For this, I will most likely need to keep a separate file to the main scene data file in the folder of each project in order to store meta-data such as date, time, owner user and commit message.

If a user decides they would not like to create separate version files at each save and would like to overwrite the data on the current file instead, they will still be

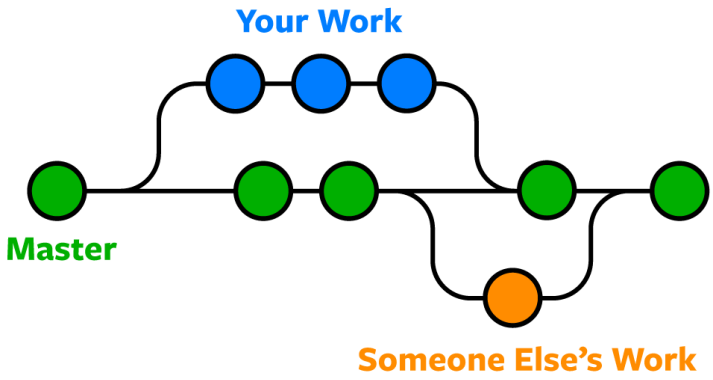
able to save time/date stamped commit messages at each save but will not be able to go back and view what the project was like at that save.

Additionally, I would like to emulate the GitHub branching feature such that if a user goes back to a previous version save of a project and then edits it, they will be given the option to create a new branch originating at the current version or “chop off” the current branch which extends from this version (and replace it with the new changes) upon saving.



This feature would compliment the merging feature mentioned earlier in 2 use cases:

- ➔ Users can create branches for experimental additions and then merge back with the main branch once they are confident that they would like to add the changes to the project.
- ➔ When multiple users are working on the same project they can create different branches and then merge their versions together at a later stage in the project. This means that teams working collaboratively can keep all their work in one project folder instead of having many separate project instances and then merging all of them together.



References

1. Blender Foundation (2019). *blender.org - Home of the Blender project - Free and Open 3D Creation Software*. [online] blender.org. Available at: https://www.blender.org/.
2. www.d5render.com. (n.d.). *D5 Render | Real-Time Ray Tracing 3D Rendering Software*. [online] Available at: https://www.d5render.com/.
3. Autodesk (2021). *Fusion 360 | 3D CAD, CAM, CAE & PCB Cloud-Based Software | Autodesk*. [online] Autodesk.com. Available at: https://www.autodesk.com/products/fusion-360/overview?term=1-YEAR&tab=subscription.
4. D5render.com. (2020). *Feature | D5 Render*. [online] Available at: https://www.d5render.com/features [Accessed 9 Sep. 2024].
5. Wikipedia. (2021). *Tyndall effect*. [online] Available at: https://en.wikipedia.org/wiki/Tyndall_effect.
6. Zero Wind :: Jamie Wong. (n.d.). *Ray Marching and Signed Distance Functions*. [online] Available at: https://jamie-wong.com/2016/07/15/ray-marching-signed-distance-functions/#the-ray marching-algorithm.
7. therealmjp.github.io. (2019). *An Introduction To Real-Time Subsurface Scattering*. [online] Available at: https://therealmjp.github.io/posts/sss-intro/.

Features & Limitations

Features

Settings Menu

Render Preferences

The “render preferences” section of the settings menu will have sliders to control preferences for the render. This will include the depth of field as well as the intensity of various attributes such as motion blur, lights and camera effects. This will allow the user to intuitively fine tune their render to achieve the look they are going for without having to know all of the exact values for the render preferences or re-type guesses until they find something which works. In practice, this helps people get renders finished quicker so they can move onto other areas of their project.

Additionally, users will have the option to toggle certain effects on and off which cannot be easily scaled. Doing this will allow users with different hardware capabilities to get scenes rendered more easily by removing some of the less impactful effects in their use case.

Finally, there will be two dropdowns available for the user to specify the desired resolution and aspect ratio for their render - changing the resolution will not affect the aspect ratio.

I do not expect this to take long to implement as once I have the basic slider/dropdown functionality which is easy to achieve with most standard ui libraries, all I need to do is create multiple instances of each of these for each preference and display them on a window. However, this feature is dependent on the implementation of the rendering functionality itself, as well as all of the effects to go with it which means I may only be able to implement it in a later sprint.

Gallery

Next, there will be a gallery section to the settings menu which will contain two subsections: a textures gallery and a backgrounds gallery. Both of these will display all the textures/backgrounds the user has downloaded, which they can import into any of their project folders, in a grid format - showing a small visual sample of each texture/background above its title.

The gallery will have a smart searching system which allows users to search for both textures and backgrounds without knowing their exact title. This will be accomplished by allowing the user to specify multiple categories which a texture/background fall into upon importation into their gallery database, these categories can then be used as search terms when looking for a specific import in order to narrow down the search. This becomes increasingly valuable as the number of files the user downloads into their gallery increases. The user will be able to access a number of materials such as glass, steel and marble by default as they can be generated without the use of texture mapping. There will be an option to export the gallery databases (either the textures, backgrounds or both) into a shareable format. When one user downloads and loads the project of another user, they will have to download the textures/backgrounds used in that scene as part of the project file, they will then be given the option to either save these assets into their own gallery or keep them just inside the project file.

This will most likely be the most lengthy aspect of the settings menu to implement but I still do not think that it will take an extremely long time as all of the file handling code will be done using python which is my most confident language.

Scene Creator

The scene creator tool is the main interface that the user will be interacting with while using the software (There will be an option to view this interface in either dark mode or light mode).

Object Manager

This will be a collapsible tab at the side of the screen which will contain a list of all the objects added to the scene, along with all of their attributes on a collapsible dropdown (each attribute will have a slider for the user to easily edit the object). Each object will have a visual aid next to its title, showing a small copy of what the object looks like.

Object Addition Menu

When the user wants to add an object to the scene, a window will appear in the middle of the screen allowing them to choose the type of object to add and change its default attributes if they want to.

Importing Assets

Users can import backgrounds/textures from their gallery into a project scene and then apply them to the scene/objects. I hope to be able to implement a drag and drop feature for this but I may need to just create a small pop-up menu window when applying assets. The drag and drop mechanic is the most common way of implementing this kind of feature so it would be easiest for less experienced users to understand. However, this is more complicated so if I am running low on time I may not be able to complete it.

I am going to have to validate imported files before trying to apply them to scenes or objects. This can either be done at the point of application or when downloading into the gallery. I think it is most advantageous to validate at the point of download so that if the user wishes to use the same assets in multiple projects they can do so without having to perform redundant validations.

Scene Previews

I would like to implement a 3D visual preview of the scene that is going to be rendered as the user is adding and editing objects in the scene. This would make it far easier for the user to see what their scene is going to look like and make adjustments without having to continually render and re-render their scene. This may be fairly difficult to implement depending on the capability of the libraries available to me but given the profound impact the feature would have on project development I think it is worth spending the time to find a way to successfully implement it.

Rendering Methods

If a user has lots of scenes they would like to render in high detail and with lots of computationally intensive effects, it may be useful to add a render queuing option which allows the user to schedule the rendering of multiple scenes one after another in a user-specified order. This would allow the user to leave the program unattended while it renders and not have to worry about coming back to start each of the new renders separately.

Textures

There are 4 primary ways which rays will be altered when interacting with objects of different materials. For opaque materials, the light will be scattered either diffusely (where each ray is given a random angle in the opposite direction to the way it intersected with the object) to give a matte texture or specularly (where the angle of incidence is equal to the angle of reflection) for reflective surfaces. Transparent materials will allow most light to pass through them with potentially some distortion or they will reflect a small amount of light back like a reflective material would (this is to make the material still visible and not look like there is simply nothing there. For example, reflections off water will be slightly stretched parallel to the viewing direction and the intensity of the light will be slightly reduced (a similar effect will be seen on metallic materials, using a blur rather than a stretch). Finally, translucent materials will allow some of the light to pass through and some light to be diffusely or specularly scattered spending on the material. One method of doing this is to have some of the light reflected off the surface and then the rest of the light diffusely scattered after they have partially passed through the surface of the material - however this method is somewhat complicated to calculate so will only be added in a later sprint if I have a successful translucency implementation already as well as spare time left over. The only other way which materials will be rendered without the use of a texture map is using random perlin noise to generate a non-repeating marble effect.

Texture mapping will be used to represent any other materials the user wishes to import and use in their scene. This, along with the generated textures allows the user to create more realistic objects and scenes as well as making the scene more interesting to look at which is important for making visual products such as games and movies more engaging for the consumer.

I do not expect the bulk of this feature to take long to implement and it is a core aspect of rendering interesting scenes so I plan to complete this during one of my earlier sprints.

Effects

Similar to textures, effects such as cinematics, volumes (e.g. fog, smoke) and motion blur make the scene more engaging to look at and can contribute greatly to the feel and theme of the image. Additionally, effects can help with the realism of a scene - making it appear more professional. These effects are as follows:

- ➔ Ambient Occlusion
  - ◆ Makes lighting and shadows smoother and more realistic
- ➔ Tyndall Effect
  - ◆ More realistic interaction between a liquid and the light passing through it
- ➔ Luminance
  - ◆ Secondary light sources
- ➔ Defocus Blur
  - ◆ Blurring parts of the image outside the depth of field to render scenes more realistically as if it were taken by a real camera.

Some of these effects are more complicated than others but many are definitely achievable so I plan to implement the majority of these after I have implemented the texture functionality.

Path Tracing

Sphere intersection and surface normals will be required to create my first ray traced images, later down the line I will implement quadrilateral intersection in order to work with more complex scenes. After this I will implement bounding volume hierarchies to support more objects in a single scene without impacting rendering times too much. This provides support for users or teams working on larger scale professional projects without hitting rendering bottlenecks - however is not really necessary for smaller scale personal projects as I am only rendering still images rather than video which does not take nearly as much time. Finally, I will implement antialiasing where an average is taken from multiple samples around each pixel in order to render smoother curves for less pixelated images.

The majority of these should not take a very long time to implement and are vital to creating a fully functioning ray tracer, so I plan to complete most of these in early stages of development and then further optimise the BVH if I have the time capacity for it later on.

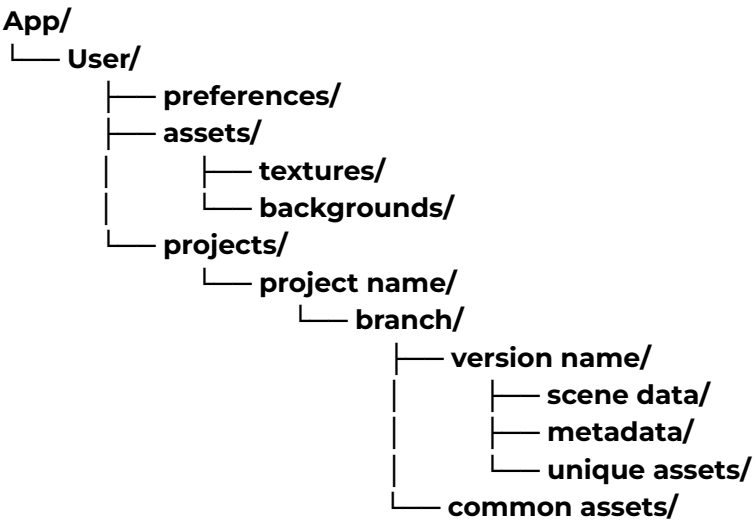
Version Control

Upon opening the program, the user will have to login to their account in order to access their projects - these accounts will only be saved locally with the purpose of allowing multiple users to use the program on the same device. When a user saves a scene (the data will be saved within their account only) they will be given the option to either overwrite the currently saved scene data or create a new version copy of the project, as well as being able to add a comment when they save to indicate the changes made at this point in their development - these comments along with a date/timestamp will be stored in a metadata file within the project folder. This allows users to decide how rigorously they wish to keep track of the project's version control depending on the scale of the project and how many people are working on it. Meaning projects of varying scale from different types of developers can be carried out using my software.

When opening a project, the versions will be displayed in reverse chronological order (with the most recent version at the top). This makes it easy to keep track of projects with lots of different versions from different points in time.

Additionally, I wish to implement a system which allows users to create version branches in their project folders which they can create separate versions of the project within. Users can then merge different versions back into a single branch. This merging mechanic may also become cross-user if I have the time but is not integral to the use of the software as people who wish to work collaboratively could just share a team account.

The folder hierarchy of this system is as follows:



I do not believe that any of this (apart from potentially the merging feature) will take very long to implement as it is mostly path and file handling which I am fairly confident in already. However, much of this is not required for the function of the program as a whole and simply just improves the user experience, particularly for larger projects and teams. Because of this, I will most likely complete this feature during a later stage of the project development process.

Limitations

My software will be made for rendering still images which means I will not be implementing support for videos or real-time rendering. These may be largely impactful additions but are primarily used for larger scale professional productions and require a lot of computational power to perform - as well as being extremely difficult to develop. These criteria do not fit the majority of my target demographic which is solo developers and small teams.

Additionally, I will not be providing support for cross-device user interaction as it would require connecting to the internet in order to login which may be complicated to set up and takes more development time away from the ray tracing itself. This means that users will not be able to edit the same project at the same time - however this, again, is usually only something needed for big teams working on larger scale projects.

Requirements

Non-Functional Requirements

Development

For the ray tracing portion of my program I was considering using either rust or c++ as my programming language due to their speed and popularity for graphics and simulation - which means there are lots of resources available to learn from when I run into issues during

development. Rust is a newer language so does not have as many resources but I decided it would be preferable due to its safer memory management. I will not be using any external libraries or tools for this part of the project.

For the user interface I will be using Python with Tkinter as the UI interacts with the version control system directly - which is something I will also be developing in Python due to my confidence in the language and its built-in module support for file and path management. I chose Tkinter as my UI library and PyThreeJS for the 3D environment, both for similar reasons as my previous choices (popularity for use cases similar to my own and prior experience).

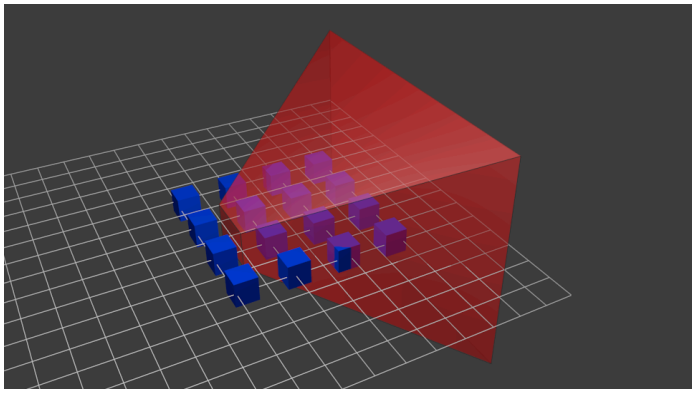
Additionally, I will be using a combination of Python and SQL in order to manage project and user data within my app. This is because I will already be using python for a large portion of my code, so it reduces the complexity of making different aspects of my code base interact, and it is easy to integrate with SQL.

I will require a computer with a graphics card in order to test my code during development due to the interaction of PyThreeJS and my rust code with the GPU.

My project has a number of different parts which depend on each other, and the potential for requirements to change throughout development. Because of this complexity I have decided to use the agile software development lifecycle, which allows me to build a basic shell for the software and iteratively build on it in layers - ensuring that all parts of the software are up to date with each other. Additionally, my stakeholders are far more experienced in 3D graphical development than myself so using a methodology with frequent user feedback helps me to handle changing requirements and ensure that my final product meets the needs of my target market - making use of quality assurance testing after each sprint.

Usage

Since PyThreeJS and my rust ray tracing program will both make use of the GPU in order to render images, the user would require a graphics card to run my software. This limits the number of devices my program can run on but is necessary to render images of high quality in realistic amounts of time because of the GPUs parallel processing capabilities.



Functional Requirements

| Requirement | Priority | Raised By | Description | Feature |

Object Manager	Must Have	Me (Jamie)	Tab with visual list of all the objects in the scene and their variable attributes on display to edit, as well as buttons to add and remove objects from the scene	Scene Creator
Positionable Camera	Must Have	Me (Jamie)	Method for user to move the camera to different positions and orientations around a scene	Scene Creator
Sphere Intersection	Must Have	Me (Jamie)	Calculating ray intersections with spheres	Path Tracing
Surface Normals	Must Have	Me (Jamie)	Calculating vectors normal to surfaces at the point of ray intersection	Path Tracing
Antialiasing	Must Have	Me (Jamie)	Taking multiple samples of a pixel at different nearby points and calculating an average colour value	Path Tracing
Diffuse Materials	Must Have	Me (Jamie)	Scattering rays randomly off objects which don't emit light in order to create a matte texture	Textures
Reflections	Must Have	Me (Jamie)	Specular scattering off light off an object	Textures
Dielectrics	Must Have	Me (Jamie)	Splitting single rays into separate reflected and refracted rays upon intersection to create transparent surfaces	Textures
Quadrilaterals	Must Have	Me (Jamie)	Calculating ray intersections with quadrilaterals	Path Tracing
Lights	Must Have	Me (Jamie)	Being able to include numerous light sources throughout the scene	Effects
Multiple Users	Must Have	Martin	Ability to sign in with different users who are working on different scenes	Version Control
Savable	Must	Me	Allowing the user to save all the data for a scene (as opposed to just the final	Version

Scenes	Have	(Jamie)	render)	Control
Object Addition Menu	Must Have	Me (Jamie)	Pop up window when adding objects to choose their attributes, the quantity of the object you are adding and the position(s) of the object(s)	Scene Creator
Resolution Dropdown	Must Have	Me (Jamie)	Dropdown to edit the resolution of the rendered image (from a range of standard options) without changing the aspect ratio	Settings Menu
Aspect Ratio Dropdown	Must Have	Jason	Dropdown to select aspect ratio of rendered image, from a range of standard options (including an option for a custom aspect ratio)	Settings Menu
Water Reflections	Must Have	Me (Jamie)	Blurring and stretching the reflections off water as well as reducing the intensity of light reflected in order to better replicate the way light bounces of the surface of water	Textures
Backgrounds	Must Have	Martin	Having a range of different types of backgrounds which can be applied to a scene for different effects and use cases	Scene Creator
Imports	Must Have	Martin	An option to import additional textures and backgrounds from your computer file system	Scene Creator
Import Validity	Must Have	Martin, Me (Jamie)	Checks to ensure imported texture and background files are the correct format	Scene Creator
Defocus Blur	Should Have	Me (Jamie)	Scaling blurring at different depths to recreate the "depth of field" effect caused by real cameras because they gather light through a large hole rather than a point	Effects
Volumes	Should Have	Me (Jamie)	Creating the effect of translucent gases such as smoke or fog within a scene	Effects
Texture Mapping	Should Have	Martin	Applying material effects to objects in the scene	Textures
Perlin Noise	Should Have	Me (Jamie)	Creating non-repeating textures such as marble using random noise generation	Textures
Instances	Should	Me	Object transformations such as rotations and translations	Scene

	Have	(Jamie)		Creator
Texture/Background Manager	Should Have	Jason	Galleries showing all of your downloaded textures/backgrounds. Glass, steel & marble will be default materials available which are generated without using a texture map	Settings Menu
Performance Toggles	Should Have	Me (Jamie)	Toggles to include or disclude effects such as shadows, metallic reflections, reflections, volumes, motion blur	Settings Menu
Metallic Blurring	Should Have	Me (Jamie)	Adding a smooth blur to the reflection effect seen on metallic objects to make it appear more natural and realistic	Textures
Ambient Occlusion	Should Have	Me (Jamie)	Calculating a scalar value for each point on a surface to determine the amount of ambient light being reflected off it from the rest of the environment. This helps to smooth shadows, making 3D objects appear more realistic	Effects
Renders Slider	Should Have	Jason	Creating sliders to edit preferences such as depth of field, when rendering so values can be adjusted more intuitively	Settings Menu
Saving Options	Should Have	Jason	Allow users to choose between overwriting the current file and creating a new file with the updated content when saving projects	Version Control
Version Descriptions	Should Have	Me (Jamie), Martin	Allowing the user to add a short comment or description to each version of a project as they save it - similar to Git's commit messages	Version Control
Gallery Searching	Should Have	Jason	Assigning multiple search terms to each texture/background to make searching more intuitive	Settings Menu
Dark Mode	Should Have	Martin	Option to view the entire interface in a dark mode	Scene Creator
Chronological Versions	Should Have	Me (Jamie), Martin	Showing the dates on which versions of a projet were saved and displaying them in reverse chronological order (top version most recent)	Version Control
Scene Templates	Could Have	Me (Jamie)	Introducing a few basic preset scenes for common use cases	Scene Creator

Cinematic Camera Effects	Could Have	Me (Jamie)	Effects such as bloom, lens flares, chromatic aberration & vignette	Effects
Visual Scene Previews	Could Have	Me (Jamie)	Low detail preview of the scene as it is being created and edited	Scene Creator
Illustrated Dropdowns	Could Have	Me (Jamie), Martin	Small diagrams next to dropdown option descriptions to illustrate the function of the option	Scene Creator
Bounding Volume Hierarchies	Could Have	Me (Jamie)	Discarding subsets of ray calculations depending on intersections with larger bounding volumes	Path Tracing
Subsurface Scattering	Could Have	Me (Jamie)	Rendering realistic translucent materials by reflecting some rays off the surface and scattering the rest after they have partially passed through the object	Textures
Merging Scenes	Could Have	Me (Jamie)	Merging multiple different scenes from the same user or across users	Version Control
Tyndall Effect	Could Have	Me (Jamie)	Increasing the visibility of light as it passes through certain fluids	Effects
Motion Blur	Could Have	Me (Jamie)	Creating the effect of movement in a still frame by adding blur to an object in one direction	Effects
Rendering Queue	Could Have	Me (Jamie)	If you want to render multiple scenes (or the same scene in various different ways), which might take some time you can create a queue of scenes to be rendered which will start one after another automatically	Scene Creator
Parallel Rendering	Could Have	Me (Jamie)	If you want to render multiple scenes or versions of a scene at once, you can select a number of scenes to render in parallel with each other	Scene Creator
Project Branching	Could Have	Me (Jamie), Martin	Allowing users to make numerous version branches from different saves along the main branch	Version Control

Cross-device user interaction	Won't Have	Me (Jamie)	Users from different devices collaborating on one project, making use of the internet	Version Control
Video Rendering	Won't Have	Me (Jamie)	The user will not be able to string together multiple versions of a scene and turn it into an animation	Scene Creator
Real Time Rendering	Won't Have	Me (Jamie)	The scene won't be rendered in real time using ray tracing as it is being edited	Scene Creator

High Level Success Criteria

1. **Settings & main menu** should be intuitive and easy to navigate
    - This is to ensure that users who are new to 3D development are still able to make use of my software and can gradually learn more advanced features

2. **Gallery** should support storing, displaying and searching plenty (>= 100) of textures/backgrounds
    - In order to support larger scale projects, as well as users who have many projects built up - it is important that frequently used assets are readily available without the user having to waste time re-importing them

3. **Rendered images** should complete in a short amount of time and be produced to accurately represent the scene
    - I have to compete with other currently available tools in order to appeal to my target market which means my software needs to produce high quality images without sacrificing time

4. Software should support **multiple user accounts** with different projects in each account and be able to quickly switch between accounts and projects
    - Many inexperienced groups of users may only have access to one machine that meets the hardware requirements to run my software, so having multiple user accounts on one device is essential to delivering my product to as many people as possible

5. **3D environment** should be visually appealing and have enough contrast to clearly see different aspects of the scene
    - The 3D environment is most likely going to be the most important feature in terms of user experience when using my software. So, especially when dealing with complex scenes, it is very important that the user can easily see what they are working on without getting lost

6. Users should be able to well manage **different versions** of a project and smoothly navigate between branches without data getting lost or corrupted
    - 3D graphical design can be a very time consuming process especially when creating complicated projects. This means it is vital for the software to be extremely safe when dealing with project data - as losing it could result in many wasted user hours

7. Scenes should support the addition and editing of **many objects** without causing excessive lagging
    - This is primarily for supporting semi-professional projects which are likely to be of a larger scale and will have to deal with the addition and editing of a wide range of different objects throughout the scene

8. **Materials** and **generated textures** should be precisely mapped to shapes/objects and accurately represent the surface
    - Inconsistent texture mappings are quite common in low quality 3D graphics and decrease the realism of the scene drastically. This means it is important for my software to display textures accurately if I want to be able to render competitive realistic scenes

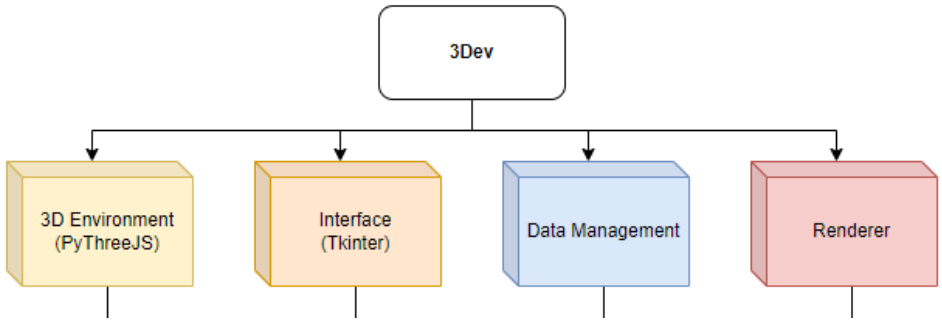
9. Users should be able to schedule and run **multiple renders** at once without having to manually start each render one after the other
    - Users may need to keep track of many different aspects of their projects e.g. in game development, which don't involve my software so this removes unnecessary time they have to spend on tasks which could be automated for them

10. **Importing/exporting** multiple assets/renders should be a swift and intuitive process which does not require unnecessary repetitive actions
    - For large scale projects, it is likely that a wide range of assets will have to be imported. Making this process as streamlined as possible gives users more time to spend actually developing their scene

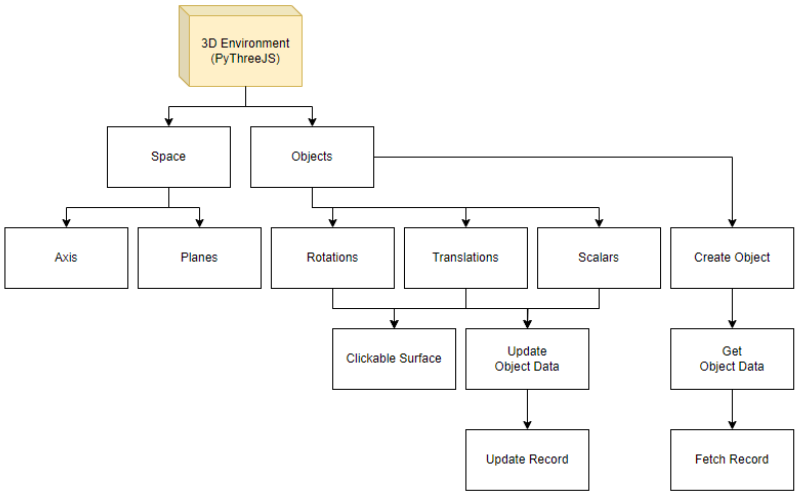
Design

Functional Decomposition

There are four main sections of development which my project will be split into - these were mentioned in the non-functional requirements section. The user interface (using Tkinter), 3D environment (with PyThreeJS) and Data Management will be coded in Python while the rendering portion of the program will be made in Rust.

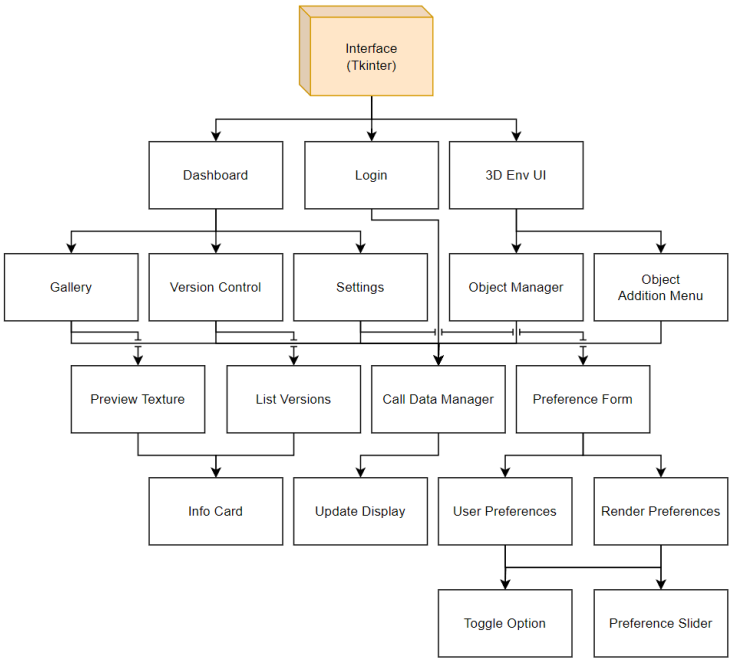


3D Environment



The implementation of the 3D environment, specifically the space portion of it, will be covered mostly under the PyThreeJS library. In terms of objects, there is some shared functionality which can be seen in the diagram above. This mostly involves reacting to how users interact with the objects in their scene and updating the object data accordingly. This means that I am likely going to need a table to store object data as the scene changes and objects are added/edited. I will decompose the data-management section of my project in order to get a better understanding for how this object data will be managed and affect other areas of the project.

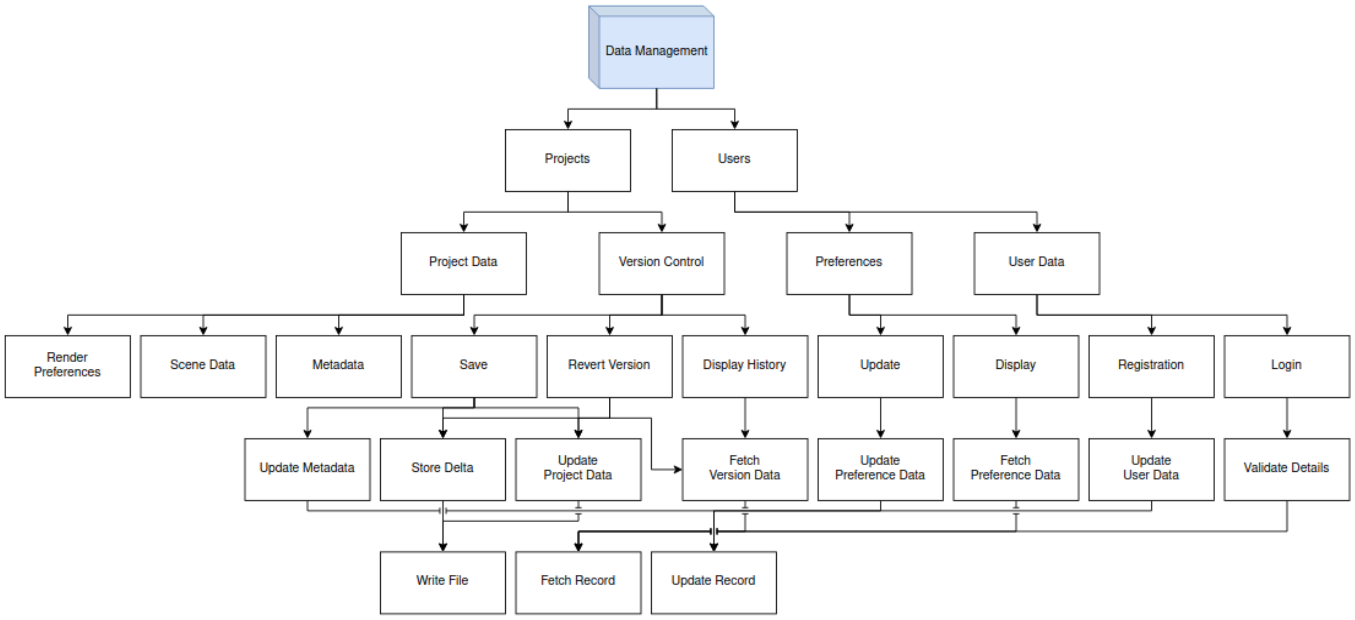
Interface



Almost all elements of my interface will interact with the data manager as the majority of the interactions the user will have with the software impact either project or user data, which means database tables need to be updated. Different menus such as user/preference forms use the same elements for representing different pieces of information - this indicates the need for a general UI module which I can call from in different areas of the program to reduce redundant code.

Decomposing the different windows I will have into their UI components allows me to get an idea for which aspects of the interface can be made modular for other functionality and which ones are more niche. I will use this distinction to prioritise working on more widely used elements in earlier sprints. Additionally, I can see that the 3D environment is going to be one of the first interface sections I will have to tackle as many of the other preferences and menus can only be properly tested based on their effect on it.

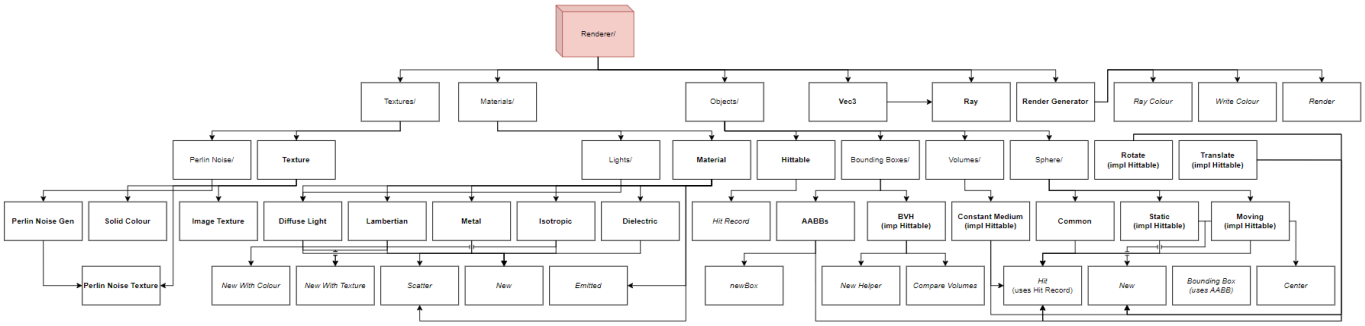
Data Management



Data management will fall under user data and project data. However, as seen in the diagram above, much of the management for project and user data involves the same subroutines. I can abstract these to make my data manager more adaptable to new tables.

Breaking down all the data I am handling into these sections gives me an idea for what tables I will need to construct my database. Viewing the connections between different branches also shows me what kind of communication will be needed between the different tables. I will use this in my sprint design to create entity-relationship diagrams for my database and also design each of my tables.

Renderer



Most of the renderer portion of my software has been broken down under either objects, materials or textures (these will have their own folders - denoted by the "/"). Within each of these folders there will be a number of sub-folders and modules (titled in bold) for different behaviours. In many cases, there are different forms of a particular behaviour which need to be implemented for different scenarios. Because of this, lots of the modules in each folder will dictate a behaviour which is a composite of the code in that file and behaviour from a higher level module. This relationship, as well as the folder hierarchy of the codebase, is indicated using the arrows. Finally, some behaviours will share different variations of the same method e.g. The different material types will all require a "*New*" method, to create a new instance of that material, specific to themselves. All methods have been indicated in my diagram using italics.

Outside of these 3 categories, I will have standalone modules for the vector calculations of individual rays, generating rendered images using all the other modules, and a vector class to help with ray calculations.

This decomposition has made it clear that the majority of my work on the rendering portion of my software will be focused on dealing with objects. This is because of the larger number of sub-branches underneath the objects section of my diagram - which shows the number of modules/sub-folders and their respective complexity (how many different methods which go into making them). Because of this, and the fact that material and texture behaviours are reliant on objects to be applied to, it is important that I prioritise getting the objects implemented quickly once I begin development.

Project Plan

I will be following the agile development methodology which breaks down the project into 3 sprints, or iterations. At the end of each sprint I will perform tests to help guide me in my next sprint analysis - so I can make informed decisions on how to prioritise my time moving forward.

Sprint 1

For sprint 1, my goal will to create semi-basic working versions of each top layer section of my decomposition - however I expect that I may have to wait until sprint 2 to have a working

version for the 3D environment as it relies heavily on input from the data manager - which relies on the 3D environment portion of the user interface. This will include features such as the **login**, **dashboard** & **object manager** (for the interface) and **path tracer**, **textures** & **positionable camera** (for the renderer). Finally, I will develop all of the data manager features which link to the ones I cover under the other sections.

Sprint 2

During sprint 2, I will be trying to complete the majority of the **3D environment** and the remainder of the other sections which don't have any outstanding dependencies. This will mean Adding in the **version control UI** and most of its data management counterpart, as well as the more niche renderer features like **motion blur** and **transformations**.

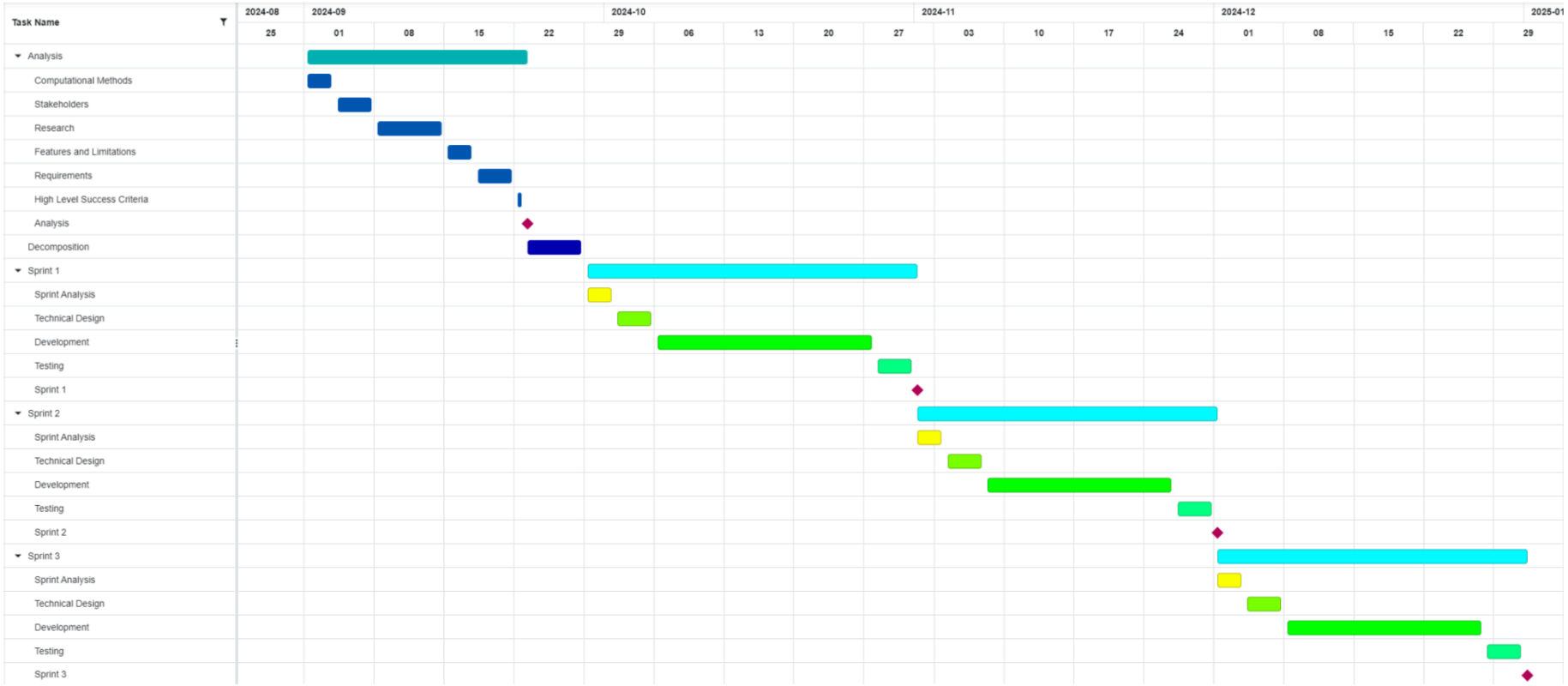
Sprint 3

Finally, for sprint 3, I plan to pull together the different independent parts of the software such as the renderer and interface into one combined program (as they will be developed and run mostly separately until this point) and update the **data manager** to work with all parts at once. Additionally, I will complete any finishing touches such as **cinematic effects** or **UI improvements** which I did not have time for in the previous sprints.

Tasks

Task Name	Start	End	Duration
Analysis	2024-09-01	2024-09-23	22 days
Computational Methods	2024-09-01	2024-09-03	3 days
Stakeholders	2024-09-04	2024-09-07	4 days
Research	2024-09-08	2024-09-14	7 days
Features and Limitations	2024-09-15	2024-09-17	3 days
Requirements	2024-09-18	2024-09-21	4 days
High Level Success Criteria	2024-09-22	2024-09-22	1 day
Analysis	2024-09-23	2024-09-23	0 days
Decomposition	2024-09-23	2024-09-28	6 days
Sprint 1	2024-09-29	2024-11-01	33 days
Sprint Analysis	2024-09-29	2024-10-01	3 days
Technical Design	2024-10-02	2024-10-05	4 days
Development	2024-10-06	2024-10-27	22 days
Testing	2024-10-28	2024-10-31	4 days
Sprint 1	2024-11-01	2024-11-01	0 days
Sprint 2	2024-11-01	2024-12-01	30 days
Sprint Analysis	2024-11-01	2024-11-03	3 days
Technical Design	2024-11-04	2024-11-07	4 days
Development	2024-11-08	2024-11-26	19 days
Testing	2024-11-27	2024-11-30	4 days
Sprint 2	2024-12-01	2024-12-01	0 days
Sprint 3	2024-12-01	2025-01-01	31 days
Sprint Analysis	2024-12-01	2024-12-03	3 days
Technical Design	2024-12-04	2024-12-07	4 days
Development	2024-12-08	2024-12-27	20 days
Testing	2024-12-28	2024-12-31	4 days
Sprint 3	2025-01-01	2025-01-01	0 days

Gantt Chart



Having already completed the analysis & decomposition section of my project, I expect to be able to complete each sprint within a month which would result in a finished product by the end of December. This means I can start the evaluation process at the beginning of January.

Development

Sprint 1

Sprint Analysis

Stakeholder Input

I got both of my stakeholders together to ask what they think is most important to the software's foundations and they provided the following responses:

- ➔ *"I think it's most important that you get a working version of the renderer complete first so you can start producing images asap, that way we have more time to test and fine tune it."* - Martin
- ➔ *"I agree with that but i'd also like some of the core interface to be added in sooner rather than later so I can get an idea for how i'm actually going to use it*." - Jason

After considering both points, I have decided to complete all of the requirements which are needed to generate my first images to a relatively high standard but without many special effects. These include most of the path tracing requirements like normals and intersections, as well as some basic texture features. In terms of the UI, I have decided to implement a few requirements involving the scene creator (as that will be interacting with the renderer) and one requirement each for the settings and version control menus.

This will give me a broad foundation to work from for the rest of the interface rather than going in detail to one specific part early on and losing sight of how the different portions interact with each other

Requirement	Priority	Raised By	Description	Feature
Object Manager	Must Have	Me (Jamie)	Tab with visual list of all the objects in the scene and their variable attributes on display to edit, as well as buttons to add and remove objects from the scene	Scene Creator
Positionable Camera	Must Have	Me (Jamie)	Method for user to move the camera to different positions and orientations around a scene	Scene Creator
Sphere Intersection	Must Have	Me (Jamie)	Calculating ray intersections with spheres	Path Tracing
Surface Normals	Must Have	Me (Jamie)	Calculating vectors normal to surfaces at the point of ray intersection	Path Tracing
Antialiasing	Must Have	Me (Jamie)	Taking multiple samples of a pixel at different nearby points and calculating an average colour value	Path Tracing
Diffuse Materials	Must Have	Me (Jamie)	Scattering rays randomly off objects which don't emit light in order to create a matte texture	Textures
Reflections	Must Have	Me (Jamie)	Specular scattering off light off an object	Textures
Dielectrics	Must Have	Me (Jamie)	Splitting single rays into separate reflected and refracted rays upon intersection to create transparent surfaces	Textures
Multiple Users	Must Have	Martin	Ability to sign in with different users who are working on different scenes	Version Control
Object Addition Menu	Must Have	Me (Jamie)	Pop up window when adding objects to chose their attributes, the quantity of the object you are adding and the position(s) of the object(s)	Scene Creator
Backgrounds	Must Have	Martin	Having a range of different types of backgrounds which can be applied to a scene for different effects and use cases	Scene Creator
Imports	Must Have	Martin	An option to import additional textures and backgrounds from your computer file system	Scene Creator

Import Validity	Must Have	Martin, Me (Jamie)	Checks to ensure imported texture and background files are the correct format	Scene Creator
Defocus Blur	Should Have	Me (Jamie)	Scaling blurring at different depths to recreate the "depth of field" effect caused by real cameras because they gather light through a large hole rather than a point	Effects
Texture Mapping	Should Have	Martin	Applying material effects to objects in the scene	Textures
Bounding Volume Hierarchies	Could Have	Me (Jamie)	Discarding subsets of ray calculations depending on intersections with larger bounding volumes	Path Tracing
Motion Blur	Could Have	Me (Jamie)	Creating the effect of movement in a still frame by adding blur to an object in one direction	Effects

Detailed Success Criteria

1. Object manager sidebar
    a. Sidebar collapses with the click of a button
    b. Object list updates in real time as objects are added/edited
    c. Objects can be collapsed/expanded by their attributes
    d. Visual object preview next to each item updates as attributes are changed
    e. Selecting object opens menu to edit attributes

2. Positionable Camera
    a. Camera properties tab in object manager sidebar
    b. Fields to enter camera properties (x, y, z, fov, focus points, depth of field)
    c. Scene rendered from updated view point once properties are changed

3. Sphere intersections
    a. Ray-sphere intersections roots calculated
    b. Ray colour updated to match sphere

4. Surface Normals
    a. Calculate surface normal vectors
    b. Shade object based on normal values

5. Antialiasing
    a. Generate pixels using multiple samples

6. Diffuse Materials
    a. Randomize vector angle off outward surface
    b. Alter number of rays absorbed based on material darkness property

7. Reflections
    a. Ray reflected at same angle to normal as it was incident on the surface
    b. Blur of reflection based on material metal polish property

8. Dielectrics
    a. Rays incident on a dielectric are split into a reflected and refracted ray
    b. The angle of refracted rays depend on the refractive index property of the material
    c. Rays incident with an angle larger than the critical angle are totally internally reflected

9. Multiple Users
    a. Page dedicated to registration/login required to be passed through before entering the application
    b. Fields for inputting the username and password

    c. Register button adds a user account
        i. Account details are added to user table
        ii. Path to new user folder (and the subsequent sub folders/files) added to system

10. Backgrounds
    a. Options for various solid colour backgrounds
    b. Gradient background function

11. Imports
    a. Asset table updated as imports are added or edited
    b. Import file formats validated before adding to file system or asset table
    c. Added imports show up as options when editing e.g. object textures
    d. Option to select or add category to place texture under

12. Defocus Blur
    a. Changing fov widens the range of depths in focus
    b. Shifting depth of field value moves this range back and forth

13. Texture Mapping
    a. Default checker texture available
    b. Applying image texture to sphere wraps it around surface

14. Object Addition Menu
    a. Object addition button opens menu to select object attributes
    b. Object added to scene once saved
    c. Listing added to object manager once saved

15. Bounding Volume Hierarchies
    a. Ray interception is calculated using the BVH and hittable bounding boxes
    b. Renders are finished quicker using the BVH method

16. Motion Blur
    a. Moving objects sampled to produce uni-directional blur effect
    b. Speed of objects (and therefore the magnitude of their blur) depends on the object speed attribute
    c. Direction of blur depends on the object motion direction attribute

17. Dashboard
    a. User should be able to navigate between menus by clicking on their title button

Test Plan

| ID | Description & Justification | Type | Required Input | Expected Output | Success Criteria |

1.11	Navigation for object manager - allows the user to open an expanded view of their objects while editing a scene	Normal	Click expand tab button	Sidebar beam smoothly expands to full menu	Object Manager
1.12	Closing object manager - allows the user to have more screen space dedicated to viewing their scene	Normal	Click expand tab button	Sidebar menu smoothly collapses to a small beam on the side of the window	Object Manager
1.2	Object manger scene reactions - keeps the object manager constantly up to date with what is going on in the scene	Normal	Add object to scene	Object listing added to sidebar	Object Manager
1.3	Object manager dropdown - allows user to see more details about particular objects in the scene	Normal	Click dropdown arrow next to object listing	Object attributes expanded underneath	Object Manager
1.4	Object listing scene reactions - keeps object manager listing attributes up to date with what's happening in the scene	Normal	Change colour of object in scene	Object preview changes to correct colour	Object Manager
1.5	Open object attribute menu - shows user detailed display of object information	Normal	Double click on an object in the manager tab	Menu appears in the middle of screen showing attributes and larger preview	Object Manager
1.61	Sidebar tab switch (OM) - lets user navigate sidebar	Normal	Click object tab button on sidebar	Sidebar switches from camera to OM tab	Object Manager
1.62	Sidebar tab switch (camera) - lets user navigate sidebar	Normal	Click camera tab button on sidebar	Sidebar switches from OM tab to camera tab	Object Manager, Positionable Camera

2.11	New viewing position - changes camera pos when new coords given	Normal	Edit camera viewing position (valid coordinates)	Scene renders from new position	Positionable Camera
2.12	Maintain viewing position - keeps camera pos when invalid coords given	Erroneous	Edit camera viewing position (invalid coordinates)	Error message: "Invalid viewing coordinates", camera remains still	Positionable Camera
3.1	Rendering sphere - makes sure sphere intersections are being calculated correctly	Normal	Add a sphere object to the scene, click render button	Rendered image of a sphere produced	Sphere Intersections
3.2	Updating sphere colour - Ensures ray colours are matching object they collide with	Normal	Edit colour property of sphere	Rendered image of sphere matching the new colour produced	Sphere Intersections
4	Surface normal gradient - Displays the mapping between surface normal values and points on the sphere	Normal	Add a sphere object to the scene with surface normal texture, click render button	Rendered image of sphere with smooth colour gradient	Surface Normals
5	Edges with antialiasing - ensures multiple samples are taken for each pixel	Normal	Add sphere to scene and position camera close to the edge, click render	Rendered image of sphere with smooth edge between the object and background	Antialiasing
6	Shaded sphere - shows how rays are being diffusely scattered off matte surfaces	Normal	Add sphere with matte material, click render	Rendered image of matte, shaded, sphere casting a shadow	Diffuse Materials
7	Metal sphere reflection - shows how rays are being specularly scattered off smooth surfaces	Normal	Add sphere with metal material next to a sphere with matte material, click render	Rendered image of matte sphere in the reflection of the metal sphere	Reflections
8	Hollow glass sphere - Demonstrates the successful reflection and refraction of rays incident on dielectrics	Normal	Add glass sphere in front of matte sphere to scene, click render button	Rendered image of distorted matte sphere seen through the glass sphere	Dielectrics
9.11	Login success - allows users to login to an existing account.	Normal	username: validUserName password: correctPassword	User successfully authenticated, Success message: "Login success"	Multiple Users
9.12	Login fail - invalid username -	Erroneous	username: invalidUserName	Error message: "username not	Multiple

	prevents users from logging into a non existing account		password: anything	found"	Users
9.13	Login fail - incorrect password - prevents users from logging into someone else's account	Erroneous	username: validUserName password: incorrectPassword	Error message: "Password Incorrect"	Multiple Users
9.14	Login fail - empty fields - stops users logging in with empty fields	No Input	username: empty password: empty	Error message: "Please fill login fields"	Multiple Users
9.21	Signup fail - invalid pass length - increases security	Erroneous	username: validUsername password: 123	Error message: "password must be at least 6 characters"	Multiple Users
9.22	Signup fail - username taken - ensures unique usernames for all accounts	Erroneous	username: existingUsername password: validPassword	Error message: "Username taken"	Multiple Users
9.3	Login button redirection - sends user into software once credentials are validated	Normal	Click login button (with valid credentials)	User redirected to dashboard	Multiple Users
10	Background updates - Ensures different kinds of backgrounds can be applied without interference with the scene	Normal	Switch between solid colour and gradient backgrounds, click render	Rendered images of the same scene with different colour/gradient backgrounds	Backgrounds
11.1	Import fail - incorrect file format - prevents users importing unsupported files as assets	Erroneous	Select invalid file format, file.invalid, to import	Error message: "Invalid format - file extension {invalid} not supported"	Imports
11.2	Import success - Ensures imported files are being added to database	Normal	Select file to import and apply it to a scene	Image rendered correctly with new asset applied	Imports
11.3	Display assets by category - Ensures assets are being assigned the correct categories on saving	Normal	Save multiple assets with varying categories and output the asset table content	Assets should be grouped into textures and backgrounds as well as divided into categories under each of those	Imports
12.11	Changing depth of field - FOV - Checks FOV is being calculated and	Normal	Render 2 identical scenes with different FOVs	Scene with larger FOV should have a larger range of depths in focus	Defocus Blur

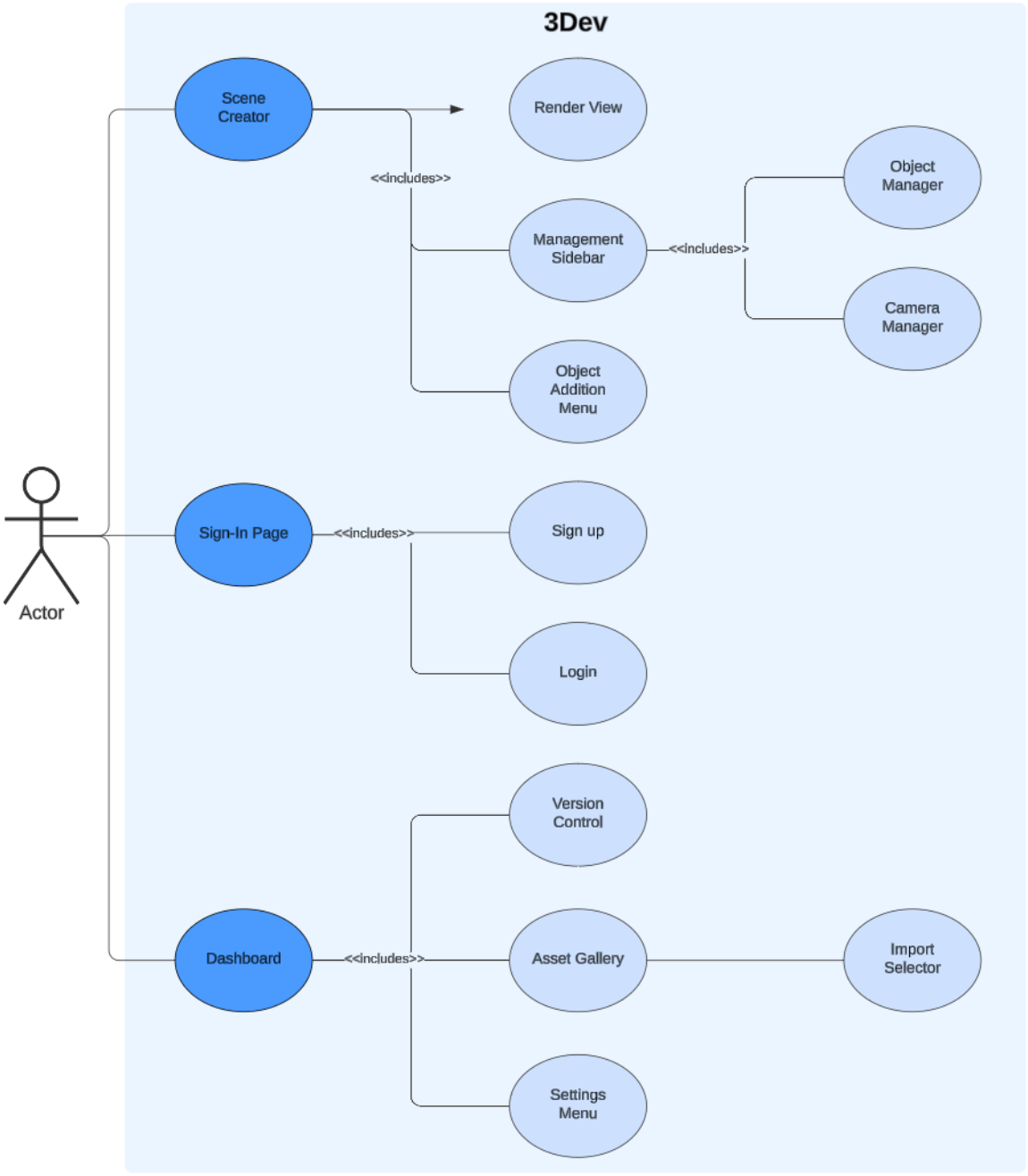
	applied correctly				
12.12	Changing depth of field - shifting - Checks the right band of depths is being focused	Normal	Render 2 identical scenes with the DOF in different places	One scene should focus on foreground while the other focuses on the background	Defocus Blur
13.1	Texture application fail - invalid texture - Prevents crashing if invalid assets are applied	Erroneous	Apply a corrupted texture to an object in the scene, click render	Error message: "{texture}, applied to {object} is invalid", the scene does not render	Texture Mapping
13.2	Texture application success - Ensures valid textures are correctly applied to objects	Normal	Apply a valid texture asset to an object in the scene, click render	Scene renders with texture correctly applied to object	Texture Mapping
13.3	Texture resizing - Ensures the application of a texture to an object remains functional as the object's attributes change	Normal	Resize and object with a valid texture applied, click render	Scene still renders with texture correctly applied and object has been appropriately resized	Texture Mapping
14.1	Object addition button - allows user to quickly add objects into the scene	Normal	User clicks addition button	Object addition menu opens in centre of screen	Object Addition Menu
14.2	Saving object success - allows scene to be updated with new objects	Normal	Click save object button (with valid attributes)	Object added to scene & object manager list	Object Addition Menu, Object Manager
14.31	Saving object fail - invalid field values - prevents user adding objects with invalid attributes	Erroneous	Click save object button (with invalid attributes)	Error message: "Invalid field value: {fieldData}", object not added to scene	Object Addition Menu
14.32	Saving object fail - empty fields - prevents user from adding objects without necessary attributes	No Input	Click save object button (with empty fields)	Error message: "Please fill in {fieldName} field", object not added to scene	Object Addition Menu
15	Rendering large scenes - Ensures optimisations meet usage requirements	Normal	Create a scene with 200 objects spaced around, render twice (with and	Scene renders in less time using BVH than without	BVH

			without BVH)		
16.11	Blur direction error - negative angle - ensures negative angles are not inputted	Erroneous	Negative blur direction attribute	Error message: "{angle} is not a valid blur direction - value must be > 0"	Motion Blur
16.12	Blur direction error - zero angle - ensures an angle is given	Erroneous	Blur direction set to 0	Error message: "{angle} is not a valid blur direction - value must be > 0"	Motion Blur
16.13	Blur direction error - max angle limit - ensures only principle angles are taken	Erroneous	Blur direction > 360	Error message: "{angle} is not a valid blur direction - value must be < 360"	Motion Blur
16.2	Blur direction success - 360 - ensures every angle can be displayed	Boundary	Blur direction set to 360	Image renders with object blur vertically down	Motion Blur
17.1	Navigation to gallery - allows users to access the asset gallery from the main dashboard and settings menu	Normal	user clicks on gallery button	User redirected to asset gallery	Dashboard
17.2	Navigation to settings menu - allows users to navigate to the settings menu from the main dashboard and gallery	Normal	user clicks on settings button	User redirected to settings page	Dashboard
17.3	Navigation to dashboard - allows users to access the dashboard form the settings menu and gallery	Normal	user clicks on dashboard button	User redirected to dashboard	Dashboard

NOTE: Every integer test ID corresponds to the number of its success criteria in the section above

Technical Diagrams

Use Case Diagram



3Dev will have 3 primary interfaces for the user to interact with - the scene creator, sign-in page and dashboard. I will now discuss the aspects of these interfaces that are going to be developed during sprint 1.

The **scene creator** is going to include the management sidebar as well as the object addition menu.

The **management sidebar** will have a section for **objects** and a section for the **camera**, both of which will contain various sliders, toggles and fields for the user to interact with.

The **object addition menu** will occupy the centre of the screen as a pop-up window when the user clicks on the object addition button, the user will be prompted to fill in fields regarding the attributes of the object they wish to add before saving and adding the object into the scene.

I will also be working on the **renderer** during this sprint which is going to be an extension of the scene creator interface as a completely separate window which opens once the user clicks the “Render” button for their scene. This has not been connected to the scene creator for this use case diagram as I will be developing the renderer completely separately from the rest of the app until the later stages of sprint two.

The **sign-in page** has two sections, **sign up** and **login**, which both require the user to fill in a username and password field. However, there are two password fields in the sign up section in order to confirm their choice, this is important because cloud or centralised data management was not within the scope of this project - meaning there is no two factor authentication as a backup in case the user unknowingly mistypes their password when signing up.

The **dashboard** has 3 sub-sections within it - version control, asset gallery and settings.

Version control will contain a list of cards which each present one of the user's projects with a title, version number, creation date and preview image - as well as a launch button to open up the scene. At the top of the page there will be a section dedicated to starting a new project.

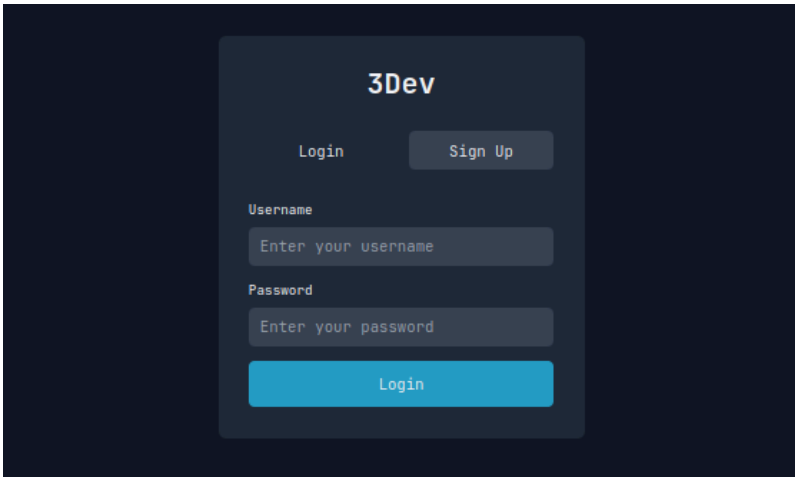
The **asset gallery** is going to have a similar layout, however, the “New Project” section will become an “Import Asset” section with a button to browse for files and an area to drag and drop them. Also, cards will be split into two sections for textures and backgrounds and laid out into more of a grid. Asset cards will contain a title, file size, import date and preview image.

Settings will be split into sections, such as user preferences and appearance, for the user to scroll through - each section will contain a variety of fields, toggles and sliders. Finally, there will be a search bar at the top for finding various assets.

I am going to create wireframes to display how I will design each of these interfaces and their sub-sections. Additionally, flowcharts and various database diagrams will help me design the logic needed to manage the information around these interfaces.

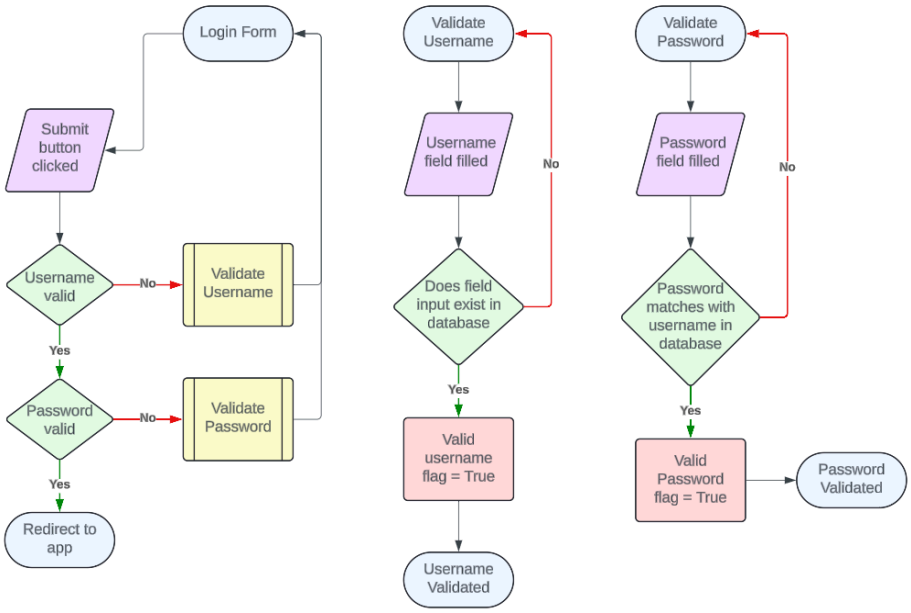
Sign-In Page

Login



This wireframe displays my design for the login page, which simply contains username and password fields as well as a “Login” button highlighted in blue. I plan to keep this blue theme throughout the design of my interfaces. Once implemented in code, the login button will change to a slightly darker blue whilst being hovered over and, if I have time in this sprint, will become momentarily smaller when clicked to indicate to the user that the click has been registered. The “Login” and “Sign Up” options are also highlighted differently to indicate which option is currently selected, however I think this current design is unclear as the Login option is the same colour as the background. This may make some users think that they are on the sign up page rather than the login page.

Aside from design, I am also going to need logic to process these interface inputs.



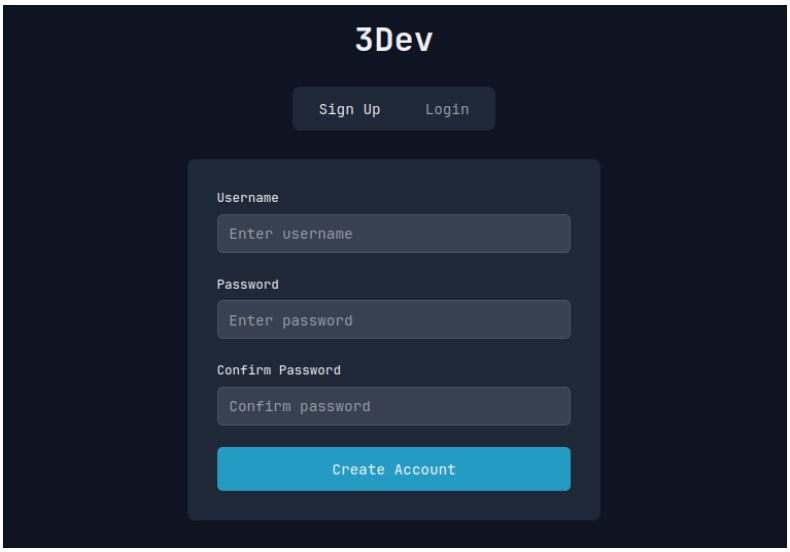
This flowchart is split into 3 sections, the main body of the program and 2 subroutines:

- ➔ **Main loop:** This piece of code will constantly be checking if the submit button has been clicked. Once it has, the “Validate Username” and “Validate Password” functions will be called - to check these credentials. These functions return a boolean value to indicate if the credential is valid. If both of these pass True then the user is redirected into the app.
- ➔ **Validate Username:** Once the username field has been filled, the program checks whether that username exists in the database or not. If it does then the function returns True and if it does not then the user is prompted to re-enter the username.
- ➔ **Validate Password:** The password validation works just like the username validation but the program checks to see if the password given matches the username.

This is the table of variables I would require to put this logic into code:

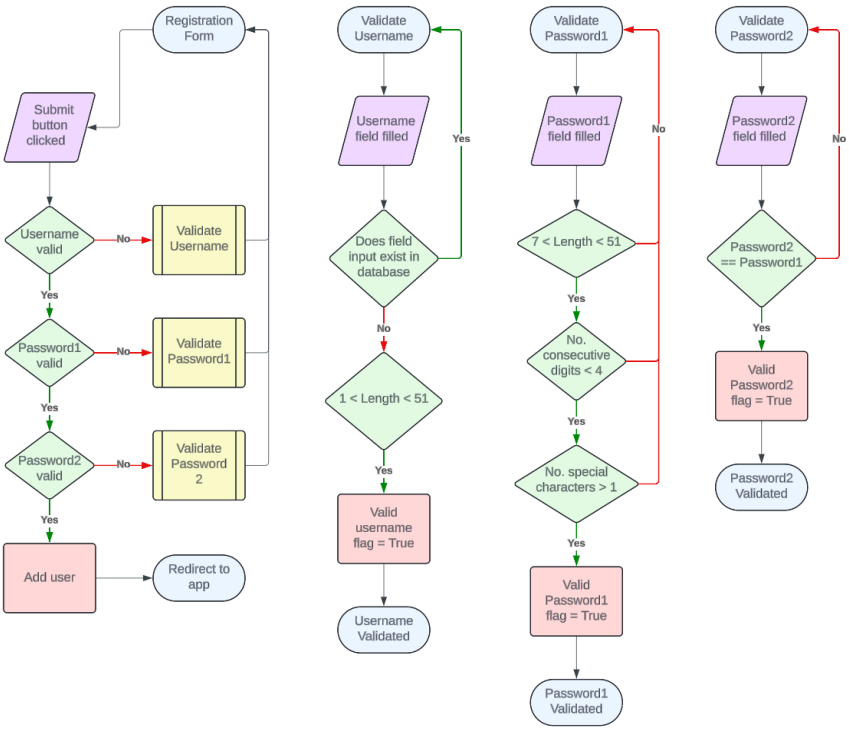
Variable Name	Description	Data Type	Validation Required
`username`	Stores the username entered by the user	String	- Must be filled (non-empty) - Must exist in the database
`password`	Stores the password entered by the user	String	- Must be filled (non-empty) - Must match the password associated with the entered `username` in the database
`is_username_valid`	Flag indicating if the username validation passed	Boolean	- Set to `True` if username validation passes (username exists in database), else `False`
`is_password_valid`	Flag indicating if the password validation passed	Boolean	- Set to `True` if the entered `password` matches the password associated with `username` in the database, else `False`
`user_authenticated`	Flag to indicate if the user has been successfully authenticated	Boolean	- Set to `True` only if both `is_username_valid` and `is_password_valid` are `True`
`submit_clicked`	Flag to check if the submit button was clicked	Boolean	- Set to `True` when submit action is triggered by the user

Sign Up



Improving on my previous design, I separated the “Sign Up” and “Login” selection from the rest of the form and decided to opt for a white text highlight to indicate which option has been selected, which I think is far less ambiguous. Additionally, I moved the sign up option to the left and the login to the right as I believe this to be a more intuitive order - since the signing up is the first option that would be picked by a new user.

Finally, there is an additional password field on the sign up page for the user to confirm their choice.



The sign up page has similar logic to the login page with a few more checks:

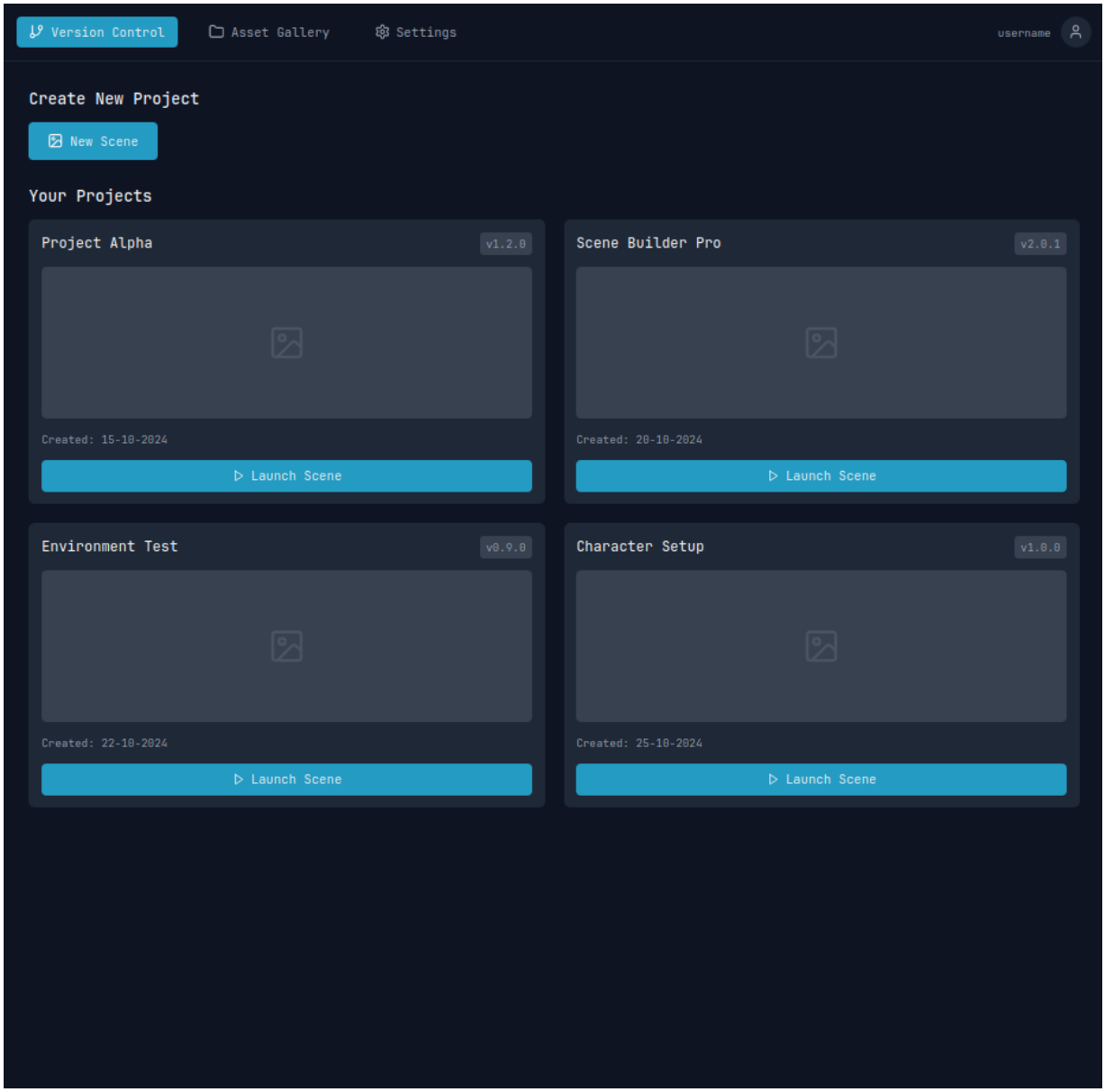
- ➔ **Main Loop:** Once the submit button is clicked, the username, password1 and password2 flags are checked - calling each of their respective validation subroutines if false. If they are all true then a new entry is added to the database - using the information submitted - and the user is redirected to the dashboard.
- ➔ **Validate Username:** This function works exactly the same as the one from the login page but in reverse, so if the username does exist then the user is prompted to choose a new one. Otherwise, the username is valid as long as it's between 2 and 50 (inclusive) characters long.
- ➔ **Validate Password1:** This subroutine checks that the inputted password is at least 8 characters and at most 50 characters long, has no more than 3 consecutive digits and has at least 2 special characters.
- ➔ **Validate Password2:** Once password1 has been validated, this function checks that the second password field matches the first.

This is the table of variables I would require to put this logic into code:

Variable Name	Description	Data Type	Validation Required
`username`	Stores the username entered by the user	String	- Must be filled (non-empty) - Length between 2 and 50 characters - Must not exist in database
`password1`	Stores the primary password entered by the user	String	- Must be filled (non-empty) - Length between 8 and 50 characters - No more than 3 consecutive digits - At least 1 special character
`password2`	Stores the confirmation password entered by the user	String	- Must be filled (non-empty) - Must match `password1`
`is_username_valid`	Flag indicating if the username validation passed	Boolean	- Set to `True` if username validation passes, else `False`
`is_password1_valid`	Flag indicating if the primary password validation passed	Boolean	- Set to `True` if `password1` validation passes, else `False`
`is_password2_valid`	Flag indicating if the confirmation password validation passed	Boolean	- Set to `True` if `password2` matches `password1`, else `False`
`user_added`	Flag to indicate if the user has been successfully added	Boolean	- Set to `True` only if all validations pass and the user is created in the system
`submit_clicked`	Flag to check if the submit button was clicked	Boolean	- Set to `True` when submit action is triggered by the user

Dashboard

Version Control

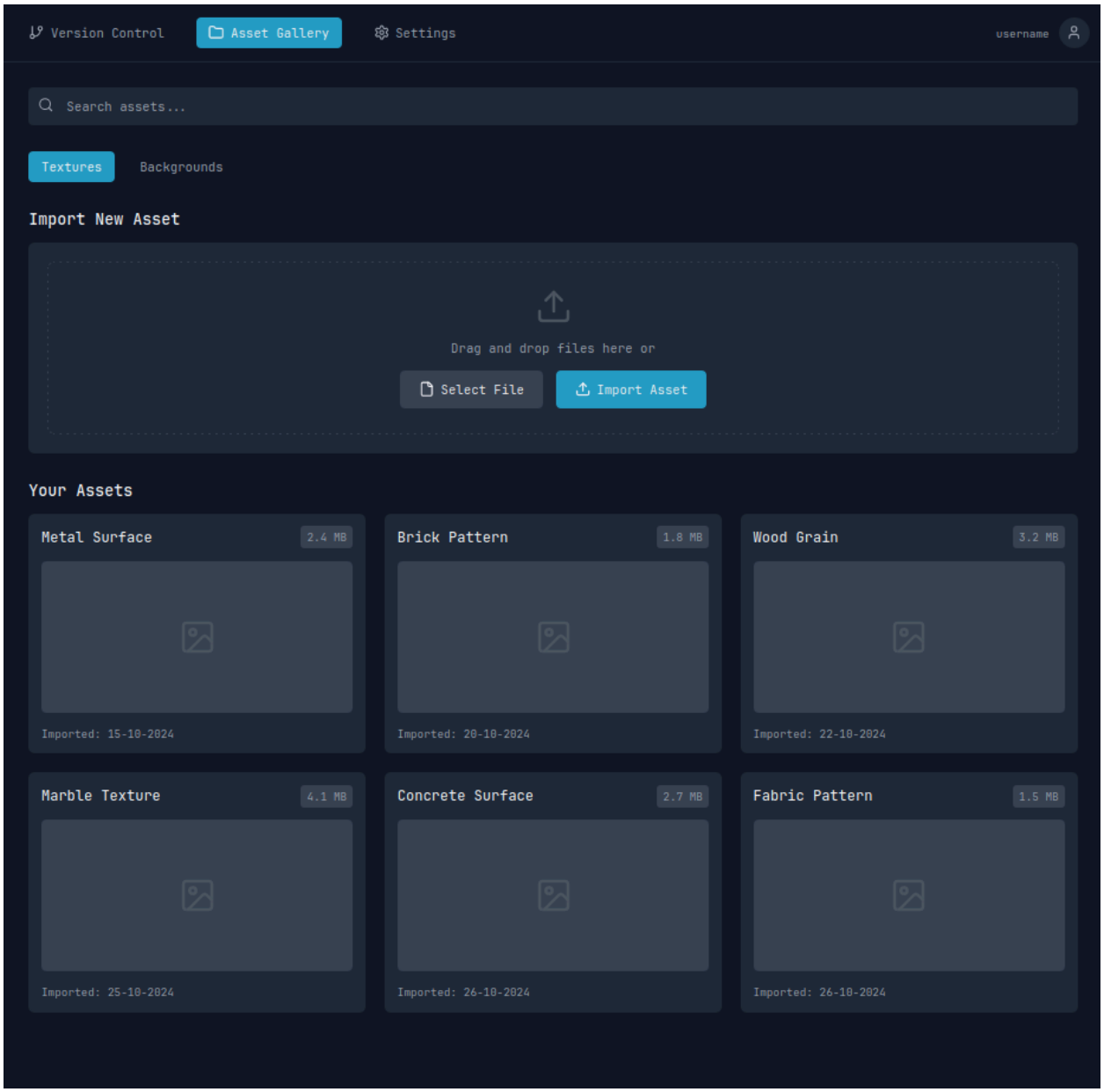


In addition to the 3 section headers mentioned earlier, I have included a user icon and username in the top right corner which will remain present on all of the dashboard pages. The user can click on this to view their user data and/or return to the login page to access a different account without the need to restart the application.

The version control page, as well as the other dashboard pages to come, shares the same font and blue colour scheme used for the sign in page.

Each project card is highlighted slightly lighter than the background colour to group the information around one project together. This makes it clear which pieces of information are in reference to which projects.

Asset Gallery

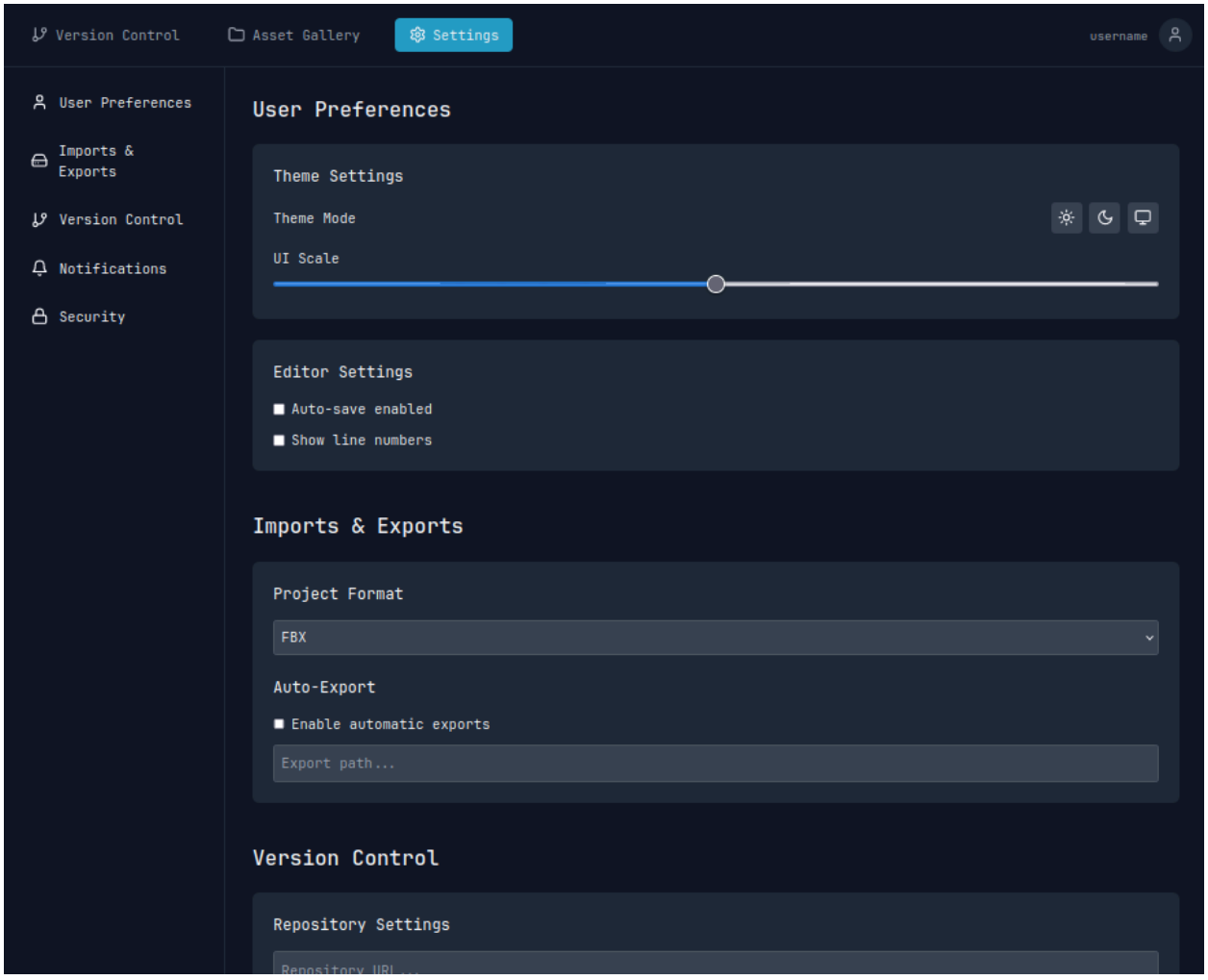


The asset gallery uses the same card format and reasoning as the version control page but, due to the likelihood of a user having substantially more assets than projects, in a 3 column grid format.

The “textures”/”backgrounds” buttons have been placed above the “Import New Asset” section so that users do not have to specify which section they are importing to for every

single asset, rather, they can remain on the textures section to import a number of textures in succession which will all be automatically allocated to the textures library.

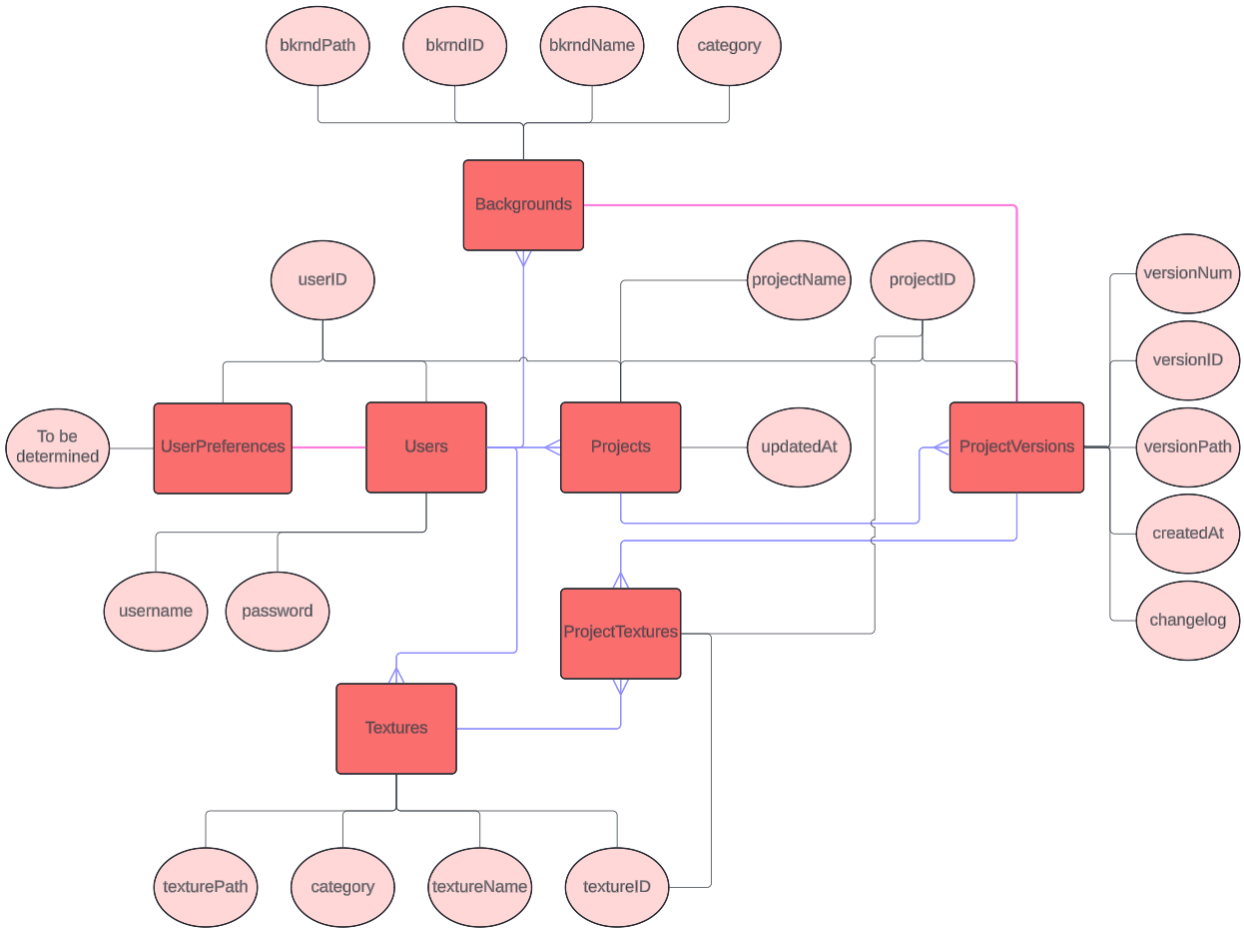
Settings



The settings menu is a continuous page that the user can scroll through to access all of the different sections, but there is also a sidebar with tabs for jumping between sections. This allows the user to decide how they prefer to navigate through their settings.

The options in this wireframe are mostly just for example of how the page will be laid out, similar to the sample projects and textures used for the other pages. Once I begin developing the code it will become more clear what preferences and options I will be able to offer to the user.

Data Management



This entity relationship diagram shows how the user and project management systems work together to manage data most efficiently. Each red box is a table and each red oval shows the fields related to it. Fields ending in “ID” are primary keys which are also used as foreign keys to connect two tables, this can be observed in this diagram when a field is connected to more than one table - such as *ProjectID* being connected to both *Projects* and *ProjectVersions*.

Pink lines between tables represent a one-to-one relationship while purple arrows represent a many-to-one relationship.

Starting with user data, I was considering keeping all core user data and user preference data in one table, since the two tables are going to be bijective anyway. However, because those sections of data are going to be used in separate contexts throughout my code, it makes it easier to keep them separate for the sake of simplifying my SQL code and making it more readable. I have not yet decided on the fields that will be included under user preference data, hence the “To be determined” on this diagram.

Asset data is split into two tables, textures and backgrounds, these tables are connected to both the user table and projects table. However, since projects can have more than one

texture and any one texture can be used in more than one project, there is an additional *ProjectTextures* table in order to avoid a many-to-many relationship. Backgrounds and textures have been separated into two different tables for similar reasons to the user data and user preferences - this simplifies the SQL when accessing the data because they are used in two different (however similar) situations.

The *Projects* table links to *Users*, *Backgrounds*, *Textures* (via *ProjectTextures*) and *ProjectVersions*. This table stores the metadata for a project which is not version specific, when the user wants to access their project, a query will be made using the *ProjectID* to find all of the version records relating to that project and pull the path from the record with the highest version number (assuming the user is trying to access their most recent version).

Programming

Testing

Sprint 2

Sprint Analysis

Detailed Success Criteria

1. Asset Manager
    b. Imported assets listed alphabetically
    c. Image preview above each asset title
    d. Search bar to look for assets (by title or category)

Technical Design

Programming

Testing

Sprint 3

Sprint Analysis

Technical Design

Programming

Testing

Evaluation