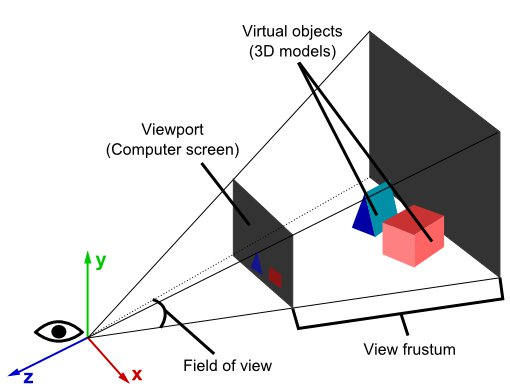
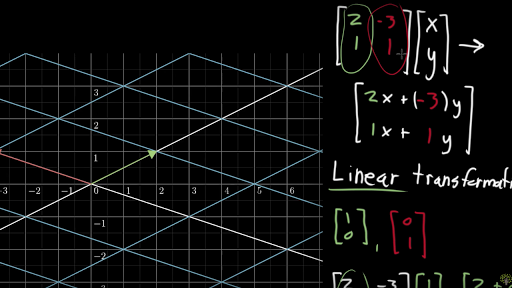
**‘3D Game Shooter’ Report**

I approached my 3D Game Shooter project knowing it will be an extremely difficult challenge and serious effort is required in 1. Learning the underlying math(linear algebra: matrices, dot products, cross products...) of 3D to 2D projection, 3D rotation, 3D movement, etc… and 2. Converting the underlying math into an efficient, runnable program using only 2 external libraries, pygame and numpy; all without copying pasting code from the internet. Although I had to spend tens of hours learning linear algebra from scratch and re-coding the entire program dozens of times to make it more efficient, it has elevated my understanding of computer science and mathematics in a profound way. Now I will go in depth about how I handled each stage of the Software Development Cycle.

**Software Development Cycle**

**Planning**

In the planning phase, I studied the feasibility of creating this program. At first I cracked the code for 3D to 2D projection, however that’s only the tip of the iceberg.

** **

My program involves rotating and moving the player’s vision freely in 3D, that was completely out of my field of expertise until I discovered linear algebra(which is taught in university). I spent tens of hours watching 3Blue1Brown and Khan Academy’s videos to gain an intuitive understanding of Linear Algebra. Once I’ve understood the math, it’s clear the program is feasible on a purely mathematical level. All I need to do left is to translate and organize the formulas into python which is in my field of expertise. Therefore, the program is feasible and we can move onto the next stage.

**Analysis**

In the analysis phase, I broke the program down to different parts and analysed the serious programming challenges. Here is the analysis for the programming challenges.

* **3D to 2D projection**
  + Figure out the math(linear algebra) formulas for 3D to 2D conversion
  + Create to2D() function in RenderUtils
* **3D Translation**
  + Figure out the math(linear algebra) formulas for translating object in 3D space + compatibility with to2D() function.
  + Add 3D translation features within to2D() function using playerX, playerY, and playerZ
* **3D Rotation**
  + Figure out 3D rotation matrices and the math(linear algebra) required to rotate points around the position of the player
  + Create transformPoints() function
  + Add rotation to points in to2D() prior to converting to 2D
* **3D Sphere scaling**
  + Figure out math to scale the 3D sphere according to how far away the player is from the sphere
  + Integrate math into drawPoints function
* **Bullet and Sphere Collision**
  + Figure out Math required to detect whether or not the crosshair is aiming directly at a sphere
  + Find a way to delete that specific sphere from the array containing all sphere objects
  + Integrate math into Sphere and RenderUtils class
* **Check if object is behind player**
  + One problem with the to2D function is it projects objects behind and in front of the player, therefore showing objects that shouldn’t be showing
  + Challenge: Since the coordinate system allows the player to freely move around and look around in 3D space, the math isn’t as simple as checking if(sphere.z<player.z) since when the player’s yaw and pitch is changed, to find whether an object is behind the player, you must check if it’s behind in respect to the plane at which is rotated according to pitch and yaw: This magnifies such a simple problem to an extremely difficult linear algebra problem
  + Solve the above problem with linear algebra
  + Integrate math into to2D() function.
* **Userinterface(UI)**
  + Create Main Menu UI
    - 4 Buttons: Quit game, Rules, Timed Mode, Harcore mode
    - Create functions for each button to do their tasks
  + Create Rules Menu
  + Create Game Over Menu
* **Game Logic**
  + Figure out functions for:
    - 2 Minute timer
    - Jump to game over menu when player loses the game
    - Initializing game
    - Random sphere generation
    - Saving High Score

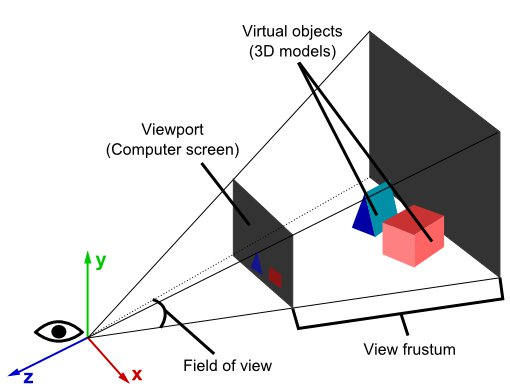
**Design**

In the design phase, I used the problems found in the analysis stage and figured out the most efficient solutions to those problems.

* **3D to 2D projection**
  + This problem can be solved using similar triangles(refer to below diagram)
  + Formula: x=(player.x-sphere.x)\*(player.fromScreenD/(player.z-sphere.z)

y=(player.y-sphere.y)\*(player.fromScreenD/(player.y-sphere.y)

Return [x,y]

****

* **3D Translation**
  + Offset the player x,y,z variables when a key is pressed
* **3D Rotation**
  + Take the dot product of a 3D rotation matrix and the point.
  + During this step, first offset the point by the player position, and after the task is completed, offset the point back. This is done so rotation is completed about the player position instead of the origin.
* **3D Sphere scaling**
  + Use similar triangles to find the projected radius of the sphere
  + Formula: radius/distance\*player.fromScreenD
* **Bullet and Sphere Collision**
  + Figure out Math required to detect whether or not the crosshair is aiming directly at a sphere
  + Find a way to delete that specific sphere from the array containing all sphere objects
  + Integrate math into Sphere and RenderUtils class
* **Check if object is behind player**
  + First rotate every point and player back to original position(prior to applying the transformation of yaw,pitch, and roll)
  + Check if sphere.z < player.z
  + Re-rotate every point and player back
* **Userinterface(UI)**
  + Create Main Menu UI
    - 4 Buttons: Quit game, Rules, Timed Mode, Harcore mode
      * Use rectangles and font to draw buttons
    - Create functions for each button to do their tasks
      * Check if mousePressed and if mouseX and mouseY is inside the button. If both true, execute tasks
  + Create Rules Menu, Create Game Over Menu
    - Create a Back button, and render text with pygame.blit()
* **Game Logic**
  + 2 Minute timer
    - Use pygame Clock and it’s clock.tick() function
  + Jump to game over menu when player loses the game
    - Create lost variable and if it’s true, render gameover menu
  + Initializing game
    - Create initGame() function, and create a for loop in range (0,30) to generate the 30 spheres
  + Random sphere generation
    - Use the timer described earlier and the random library to generate a new sphere every 2 seconds
  + Saving High Score
    - Create a highScore.txt file and store information there
    - If score is higher than high score, set highscore to score

**Implementation**

Every programmer knows your code doesn’t always go as planned, therefore in my implementation phase, I had to make serious modifications to the design of my program. Here are the major changes and modifications to the previous design.

**Major Changes**

* 3D Rotations no longer rotates the player, instead it rotates the points around the player.
  + This is done since previously, to find if a point is behind the player, you must rotate everything in parallel to the z-axis, and then compare the z coordinate and player.z. This is extremely inefficient since you rotate all points 2 times. Therefore instead, keep the player’s vision always parallel to the z-axis. When you “rotate the player”, you just rotate all the points around the player the opposite way. This change also allowed the next major change on this list since instead of rotating the player, you rotate the spheres.
* Instead of handling all the 2D and 3D transformations in the RenderUtils class, I decided to handle it directly in the Sphere class. This is done so when you rotate all the spheres, you can just type: Sphere.rotateAll(...)
* Created ScreenType enumeration class: Because using an enumeration for the screen is much more intuitive and easy to read than setting variables for if player lost. This way all you have to do is player.screenType = ScreenType.INTRO and it will display intro screen, if you set it to GAME it will display the game screen and so on.
* I realized my program does not sort the spheres, therefore when spheres are behind each other, sometimes they still render. That’s why I implemented a sortSpheres() function to sort the spheres according to their distance away from player, so closer spheres render last, adding the feature of depth.

**Minor Changes**

* Created button class inside RenderUtils: Buttons are extremely annoying to create if done individually using rectangles and fonts. Especially this way you must recreate the click detection method for each button. Therefore, I created a generalized class for button so to create a button, all you must do is write: button = Button(parameters go here...), and to draw a button, write: button.draw().
* Created a CONSTANTS class: I realized there’s a need to share constants between the Main class and RenderUtils class. One way to do this is to put everything in the player class however it seems extremely unintuitive to have constants such as the sound-files, and font-files in the player class, therefore I created a separate class called CONSTANTS.

**Self-Evaluation**

I am extremely satisfied with how my program turned out. I solved every problem in the most efficient way possible(to my knowledge) especially with the help of object-oriented programming. Although I’ve faced many difficult challenges(described above), my interest in computer science and my effort in completing this project got me to overcome them. I demonstrated:

* Excellent organization and time-management skills:
  + I divided the program into different parts and figured out the math to each part before putting it into code. Then used both class-time and time at home to complete the program
* Exceptional initiation, independence, and self-regulation skills:
  + I decided to go outside of my comfort zone and tackle the difficult problems of 3D to 2D projection. I learned linear algebra(university level course) completely from scratch by watching YouTube and Khan Academy, this was all done during my free time.

**How can I improve if I could do the project again?**

To my knowledge, there’s no way to optimize the program for extra efficiency as I’ve analysed every possible solution known to me and picked the best one to each problem. However, I could improve my time-management skills. One bad habit I had while programming was not spending enough time designing the full software in the early stages. Although I solved all the big problems, the small problems such as showing different screens was not considered. Therefore it led to wasted time in the implementation process when I had to redesign many functions to keep it compatible with the program. If I could do the project again, I will first figure out the solutions to every single problem, no matter how small it is. And then create a complete blueprint in pseudocode, then finally translate it into python.

**The END**