# Monday

## 6:40-6:55 Fill out today’s work log

## 6:40-8:30 Fill out previous sprint’s work log and learning log

**7:50 Write learning log**

## 9:20-9:40 Meeting with Bojan

## 9:40-10:00 Update work log and learning log based on Bojan’s feedback

## 10:00-16:00,19:00-20:00 Read chapter 5 of Fund. of CG

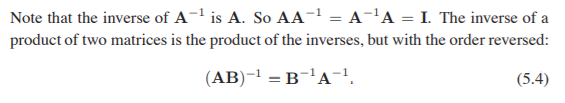
**10:00 Determinants**

Think of determinants as another way to multiply vectors. If 2 vectors create a parallelogram, the area is the determinant. If a and b are right handed, the determinant is positive.

According to the text, both areas are the same, but I can’t see how I am supposed to know that.

After looking at the situation in 2D, it became very obvious.

**10:50 Matrices**

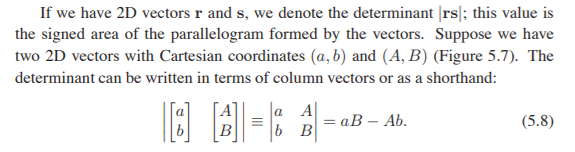


I’m not sure why this is reversed. I just read that multiplication between matrices is not commutative.

Found it out. Basically, if the order is not reversed we can never know the answer.

Matrices that are diagonal only have values in the diagonal. Symmetrical matrices don’t change when transposed. Orthogonal matrices are matrices which columns are unit and only specified in one row.

**11:55 Computing with Matrices and Determinants**

****

I don’t understand this equation, why is the matrix equal to aB - Ab? It made sense after reading chapter 5.1 again.

In algebra we use Laplace's expansion. We take the cofactor of a matrix element, which is the matrix without the row and column of the element. The sign of the cofactor is positive if the sum of the row and column are even and negative otherwise. The determinant is the cofactor times the element value.

**13:00 Implement Laplace extension**

**15:15**

The whole time I was implementing this incorrectly. Implementing this recursively is probably not even efficient but now that I know how to do it correctly. I will just try it again

**15:45**

I have decided to drop the [recursion](https://docs.google.com/document/d/1HEvP63eb50qx507pk33z1qs0v-VLCXPq05wmsVIU8OM/edit). I have already wasted too much time on this.

**16:00**

I have finally fixed the determinant. The reason why nothing worked was that I thought that the negative or positive sign is based on the matrix that is being modified.

**19:00 Computing inverses**

I may be tired but I'm struggling very much with linear algebra. It's probably because it is all new to me but the whole concept of matrices don't make sense to me. The way a determinant is calculated or why we would transpose a matrix. I will search sources to build a foundation for linear algebra because the book seems beyond me.

## 16:00-17:00 Help Brian with plane intersection and Bas with refraction formula

## 20:40-22:00 Build a foundation for linear algebra

Matrices are starting to make sense. This [article](https://betterexplained.com/articles/linear-algebra-guide/) helped me immensely with understanding what matrices are supposed to do. I thought that matrices were a way to save positional data of something. But it is a way to transform something. It is a function, it is like a spreadsheet. Here are my [notes](https://drive.google.com/open?id=1GekxCOxOUa4VmOLKlgNnCc5UAD3NQnDi). At the bottom, I have created a matrix that calculates the price of every food item and the total price that has to be paid. While it is a very simple thing, it has made me understand how the data in matrices relate to each other and how to interpret one.

**21:40 Read** [**this**](https://betterexplained.com/articles/matrix-multiplication/) **article**

I haven’t really learned anything from this article honestly.

# Tuesday

## 10:30-10:40 Fill out today’s work log

## 11:20-16:05, 16:20-19:00, 21:00-23:00 Research matrices

**11:20 Continue reading Fund. of CG Ch 5.3**

I don’t understand paragraph 5.3.1. It is about how to calculate the inverse of a matrix. I understand the concept that every value in the matrix becomes the determinant. If we divide by the determinant, we end up with the identity matrix. I don’t understand the observations that the author makes about this though. I don’t understand why the cofactors are used which is partly because I don’t know what cofactors are. I also don’t know why we transpose the matrix. I will research all this now.

**11:40 Read Math for 3D game programming 3rd ed p40 3.3 matrix inverse**

We can’t inverse every matrix. Those we can’t calculate an inverse of are called singular, this happens when one of the rows or columns consists of only zero’s (I think that means that the area would be zero which means that the determinant is zero).

The book explains everything through theories. I don’t know any of the previously mentioned theories so this is not the right way to approach this problem. I’ll try to refer to scratchapixel or khan academy.

**12:00 Finished matrix introduction on Khan Academy (pre calculus)**

I’ve learned quite a bit on what matrices are and what they do. While the tutorials didn’t give me a definition, it did explain how to do things in simpler terms. The book escalates quickly, these tutorials are more nicely passed and more thorough. These are all my [notes](https://drive.google.com/open?id=1_FsLYpG_nb6MOpGRMvoM_Z75IP5v5BQS).

**14:20 Follow a bit of the course linear algebra on Khan Academy**

**15:35 Learned how to create a linear transformation**

I know understand the definition of a transformation matrix. There are my [notes](https://drive.google.com/open?id=1TIgTgKaj3FYUhpj_EN0iaODb9bYxroNW).

**16:20 Research about formula notations**

I’ll read upon notations of formulas so I can understand them when I’m reading about definitions.

**17:00 Research the transformation matrix**

Notations are not that important right now because they are not in the ILOs. I will focus my attention on implementing the math library.

**18:20 Create own version of the rotation matrices**

[Notes](https://drive.google.com/open?id=1ktCT53UNQrzQFaOoA1SLNUKxIaBgyLGT). I’m quite proud of myself right now. I finally understand how to interpret matrices. During the unit test, I got a few failed attempts, but I was able to draw my matrix on paper and see what result there was supposed to be. I also finally understand the use of a unit circle. I will now focus my attention on researching the way determinants are calculated and how to calculate the inverse. After that, I will try to finish the unit test.

**21:00 Research method to calculate the inverse of a matrix**

[Notes](https://drive.google.com/open?id=177BBoo5XxmppeSfW4cLZP62roAZ1wHBA). While it is not really necessary to know how to do this by hand, it is interesting to know how to derive the inverse matrix. Knowing that I can get the inverse of a matrix by using the transformations to make it an identity matrix on an identity matrix is quite insightfull. I’ll be paying attention to the determinent now.

**21:45 Research determinant**

**22:15 Created proof of a 2x2 matrix determinant**

Here is the [proof/definition](https://drive.google.com/open?id=1zBzSdK0OuZp2xKQkifRJEnLyDORBV8Z_). I understand how the formula to calculate the determinant of a 2x2 matrix works now. I also understand how to apply it to get the inverse of a 2x2 matrix. I will now learn how to use this for matrices in a higher dimension.

**22:30 Implement determinant recursive**

Now that I have had all of the previous information with me, I feel like trying to implement the determinant recursively again. I now know what to do exactly.

# Wednesday

## 8:25-8:50 Fill out today's work log

## 8:50-11:50 Research how to calculate the inverse of a matrix

It's amazing to read the book again. I understand the formulas now and it’s so obvious. I’ve been calculating the inverse almost the whole time.

A-1A = I

I = 1/Det(A) \* [d -b; -c a]

A-1 = 1/Det(A) \* [d -b; -c a]\*A

See my [notes](https://drive.google.com/open?id=1zAKjfKeNUSUs70Y1CbDdwzLiSZL8UopD) for the original calculation.

I first have to change the recursive function for the determinant a bit. We need to get the determinant of every element.

**10:10**

I think that I’m actually wrong. We need to get the comatrix of every element. I’m not sure what the comatrix is supposed to be but I thought that I had to take the determinant value of every element. Reading this [article](https://en.wikipedia.org/wiki/Adjugate_matrix) I will try to implement it that way. I will also gloss over the book once more.

**Cofactor = determinant of subsequent matrix multiplied by a sign.**

**Determinant = all elements in row or column \* it's cofactor.**

We'll need to make a function that calculates the cofactor. I'm not too sure what it represents, but to get the determinant, we have to multiply the cofactor with its element. I initially mixed these operations, but to calculate the inverse, we have to split them.

Cofactor looks like this. Say we have a 4x4 matrix and we want the cofactor of 0,0

i+j%2 == 0 sign is positive

Create n-1 matrix of n matrix w/o r i and c j

Get det of n-1 matrix.

The det function looks like this

For every element in column of matrix

Det += element \* cofactor of that element

Return det

## 11:50-12:20 Implement inverse and determinant

I got the determinant working. I now have to fix the inverse.

**12:00 I’ve implemented inverse matrices with recursive functions!!!**

The functions are so simple but they are very inefficient. The determinant is as follows

Lets say n = 4

N calls function 4 times

Then the 4 times call the function 3 times

I’m not sure how to make up Big o notations so I will observe my algorithms another time. For now, I will research the lookat matrix and projection matrix.

## 13:50-16:30, 18:15-18:50, 19:45-20:20 Research what a lookat and projection matrix is

I think that the information lays in chapter 7, viewing transformation.

The viewing transformation is responsible for mapping 3D points in scene to 2D in image space. We do this most of the time by multiplying different transformations. The camera (which decides the orientation/pose), projection(which project camera space so that -1 and 1 are only seen) and viewport transformation(which maps the image size to screen size).

**14:25**

I can’t really find anything in the book about the lookat function, that is why I will implement it following scratchapixel’s [article](https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/lookat-function).

Read [coordinate systems](https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/geometry/coordinate-systems).

Nothing that special. I now know the definition of a coordinate system but I fail to see why someone would use different coordinate systems in the same coordinate space.

**14:50 short break cuz I keep falling asleep**

**15:15 continue reading the article**

Read [lookat](https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/lookat-function)

Redirected me to article about [generating view rays](https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-generating-camera-rays/generating-camera-rays)

**15:40 different coordinate systems**

I don't understand the concept of mapping world space to camera space, for a ray tracer, the orientation of a camera and casting rays through the image plane should be enough. Coordinates are implicitly converted to screen space. This is probably because SFML manages all the transformations for you. I just render to an image and draw it on screen. The image is defined in pixels which I can change the color of. Scaling all happens for me.

**15:55 plan of approach**

I'll research first what a camera is responsible for. After that. I can research the look at matrix. This matrix appears to make the camera point at something. Finally, I will look into the projection matrix which should be responsible for mapping camera space to normalized space but I am not sure. Because of deadlines, I will try to do this all today. This is not as important as spatial data structures but these are concepts that I should understand because I will use them in future blocks. I'll also have to watch animation for the PDP. I'll fit that somewhere.

**18:15 Read scratchapixel** [**viewing ray generation article**](https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-generating-camera-rays/generating-camera-rays?http://www.scratchapixel.com/lessons/3d-basic-rendering/computing-pixel-coordinates-of-3d-point?)

To calculate the position of a point in the center of a pixel, we need to **convert pixel coordinates (raster space) to world space**. We need to find a relation between the coordinates of the pixels in **raster space** and the coordinates of the same pixels but expressed in **world space**.

We have to get the pixel position **(raster space)** and transform it to **NDC** (Normalized Device Coordinates) space. After that, we have to map it to a range [-1,1]. We also have to make sure that the x and y are directed in the right direction.

We take the pixel position and divide it with the image plane size. This gives us a normalized value, but the value has a range of [0,1]. We need to map this to [-1,1]. To do this, you multiply by two and subtract 1 but you have to flip the y because it points down. Now you are in **screen space**.

If our image plane isn’t square, we can’t use range [-1,1] for the screen space. Having more x pixels will cause the image to appear shrunk along with the x-axis. We can fix this by calculating the x value in the screen size times the aspect ratio. The aspect ratio is width/height.

We can also define te FOV. Because the image distance is 1, by imagining a triangle with the y in screen space (y is in range [-1,1]), we can see that the tangent of the two sides (1 and 1) gives us tangent 1 which is 90 degrees. By increasing this angle, thus increasing tangent, we are increasing the length of Y. We can use this scale for the x and y pixels in **screen space**. After these transformations, you can say that we are now in **camera space**.

After that we apply a camera to world transformation matrix. This matrix can be applied to the camera points or to the final vector. I prefer that. This matrix is probably just a transformation matrix. Let’s design a camera class

I have a picture here with every coordinate system and their transitions.

## 20:20-23:40 Create a camera class

I’ll first make the camera class by just implementing raster space, NDC space, screen space and finally camera space. The camera space will be implicitly mapped to the world space by defining the image plane at -z, the camera will face in the -z to stay consistent with standards.

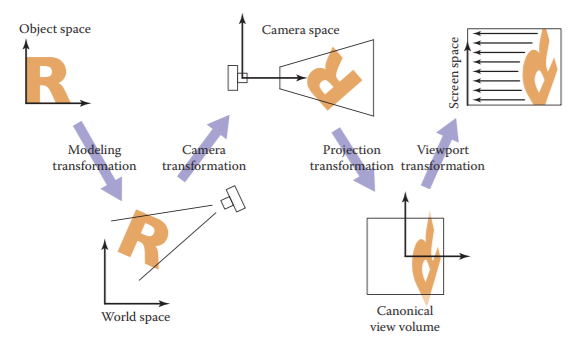
# Tuesday

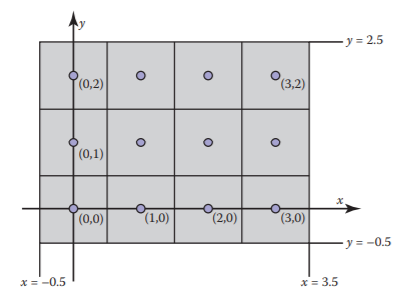
## 7:45-7:55 Fill out today’s work log

## 7:45-8:05 Fill out yesterday’s evidence

## 8:10-8:40,9:30-10:00 Research viewing transformation

I will first research the book and if I don’t understand something I will research this [article](https://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix). I’m a bit surprised about Fund. Of CG because they handle the space conversion differently then what I do. In the book they show translations from world space to screen space but I go the other way around. Maybe I am misinterpreting the book? Maybe this is for rasterization?

****

**9:30 viewport transformation**

The viewport transformation is a transformation that transforms point from NDC to screen space. NDC space has a domain of [-1,1] and [-1,1]. Apparently the book says that the NDC is a cube and also has a Z. The NDC is basically a cube that contains all the points whose Cartesian coordinates are in between [-1,1] for all axises.

Screen space is seen as pixels. The book says that the pixel position in 2D are places as integer positions. We see the pixel grid as actual coordinates. This means that the domain is [-0.5,nX - 0.5] x [-0.5,nY - 0.5].

If NDC is [-1,1] and screen space is [-0,5, nX -0.5], we need to scale by NewSize/OldSize and translate by offset.

## 10:00-10:20 Presentation code quadrants

You have a core quality like modesty,

But the pitfall is that you become invisible, your opinion doesn’t matter anymore,

So to challenge is to be more expressive.

Most of the time, you will be allergic of people that have the core quality that is your pitfall.

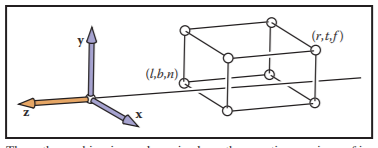
So if somebody irritates you, try to look at his pitfall and see what the core quality is to improve productivity.

## 10:00-12:10 Research viewing transformation

**10:00 viewport transformation**

The viewport is responsible for transforming points in NDC space to screen space. As discussed before, screen space has a little offset for the pixels. NDC space is a cube with [-1,1] as range for every axis. I don’t really see the need to have a viewport transformation, but have researched it nonetheless.

**11:10 projection transformation**

The projection matrix transforms a view volume (camera space) into NDC space. We can define a view volume as follow.

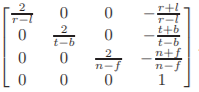
x = l ≡ left plane,

x = r ≡ right plane,

y = b ≡ bottom plane,

y = t ≡ top plane,

z = n ≡ near plane,

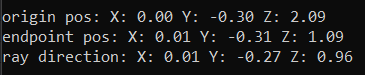
z = f ≡ far plane.

Of course, to scale one cube into another, we just use a matrix.

**12:00 Realization**

After researching and finding [this article](https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-generating-camera-rays/standard-coordinate-systems) I finally understand why I was so confused. My initial thought was right, the projection matrix is used for rasterization and not for ray tracing. I got very confused about that. While I want to learn about rasterization time is quite limited. I will not implement the projection matrix because I do not find is that important as implementing cubes.

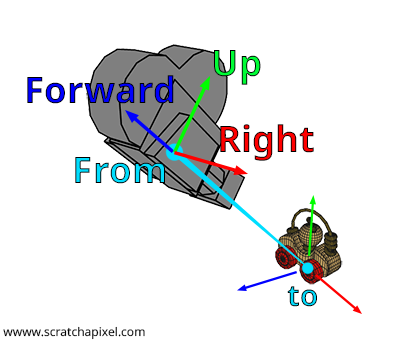
## 12:00-13:00 Debug camera class

There is something wrong with this. The ray direction should be negative. 1 - 2 = -1 but it is positive. This happens when endpoint z becomes positive.

It was a misasumption on my part. I transform the two vector begin and endpoint. To get the direction, I added the endpoint to the begin point. This is not necessary because the transformation keeps them correct with each other.

## 13:00-13:30 Help Bryan with reflections and Bas with matrices

## 13:30-16:40,18:40-18:50 Implement look at matrix

The look at matrix does what it is called. It places a camera at a point and makes it look at something. Implementing this is quite simple.

We define the camera with 3 directions. Assuming a RHS coordinate system. Up for the y axis, right for positive x axis and forward for positive z axis.

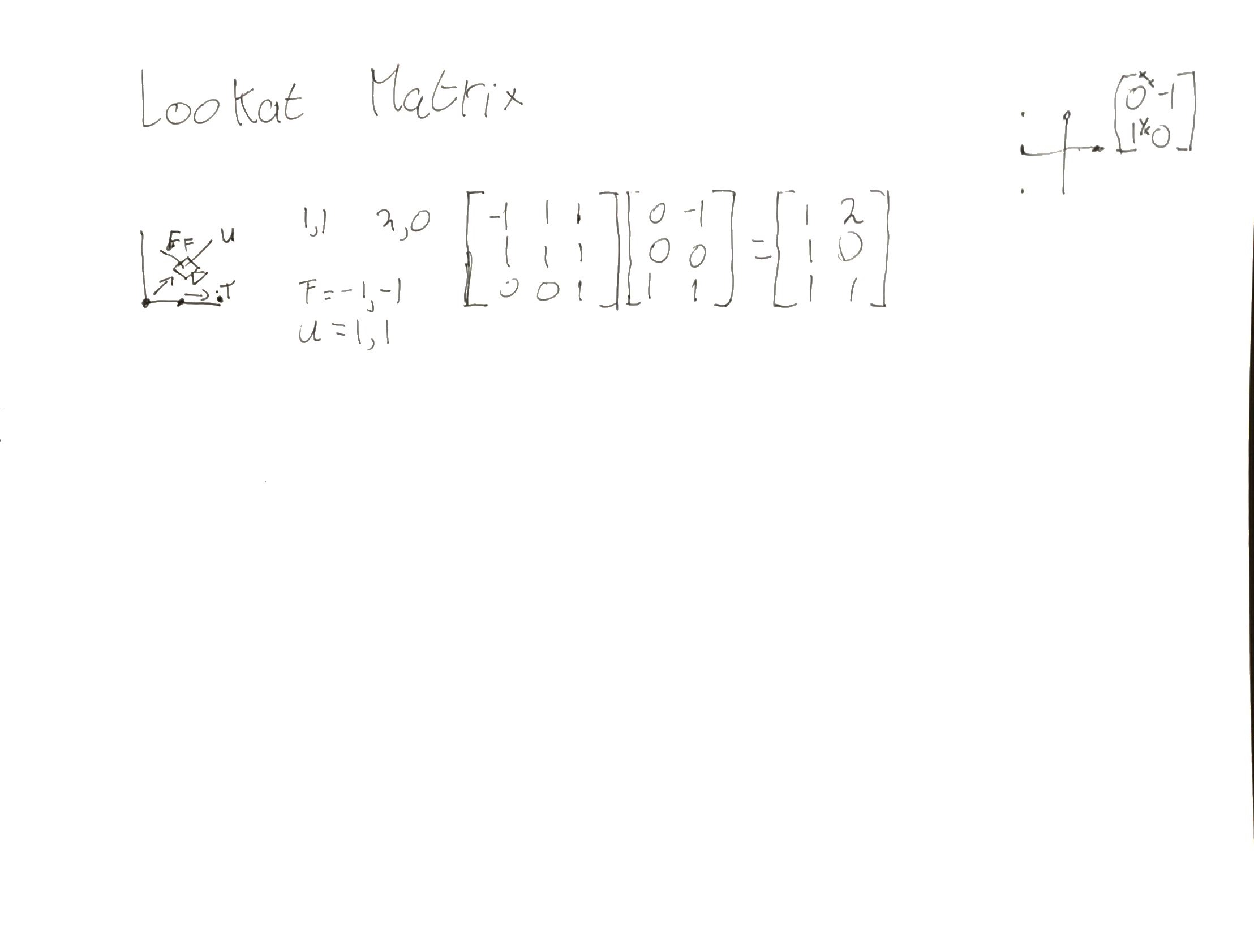
We compute the forward vector by normalizing the vector given by from - to.

We compute the right vector by getting the cross product of the forward ray with an arbitrary ray that lies in the Forward Up plane. Make sure that right will point to the right.

To calculate the actual up vector, we get the cross product between forward and right.

We then fill the 1st column with the corresponding right values. We fill the 2nd column with the corresponding up values and lastly the 3rd column with the corresponding forward values.

**14:30** **Proving why we fill values inside of matrix**

[](https://drive.google.com/open?id=1QoSS_SScSh5wewBTl314lA1hyELGidfM)

The camera is located at 0,0,0. We translate it by the point that we give to the camera. This translates all our points to the right location. After that, we have 2 vectors (3 normally) of unit length. They represent the offset that the direction will have.

**14:45 Implement lookat matrix**

My 3th column is inverted. I’m a bit confused now because I always interpreted matrices as row major but apparently row major matrices have the translation in the 4th row. But if you want to use that transformation matrix, you have to use a row vector.

For some reason the lookat is in row major. This is not that odd because the matrices are also in row major, but the translation tests are in column major. I’m not sure why they aren’t consistent, but I will implement the lookat function as a row major because I’m not sure if I can modify the unit test.

On second thought. I will fix the unit test, I will leave the old one there, but implement it how I would have checked it.

**16:10 Testing the lookat matrix**

I don’t think that it is working as intended. It doesn’t keep looking at the location that it has to look at. I’ll look at my implementation again.

**18:40 Problems**

I’m having trouble with finding a good source of information to make this lookat matrix. I’m considering thinking up one myself because scratchapixel’s solution didn’t work.

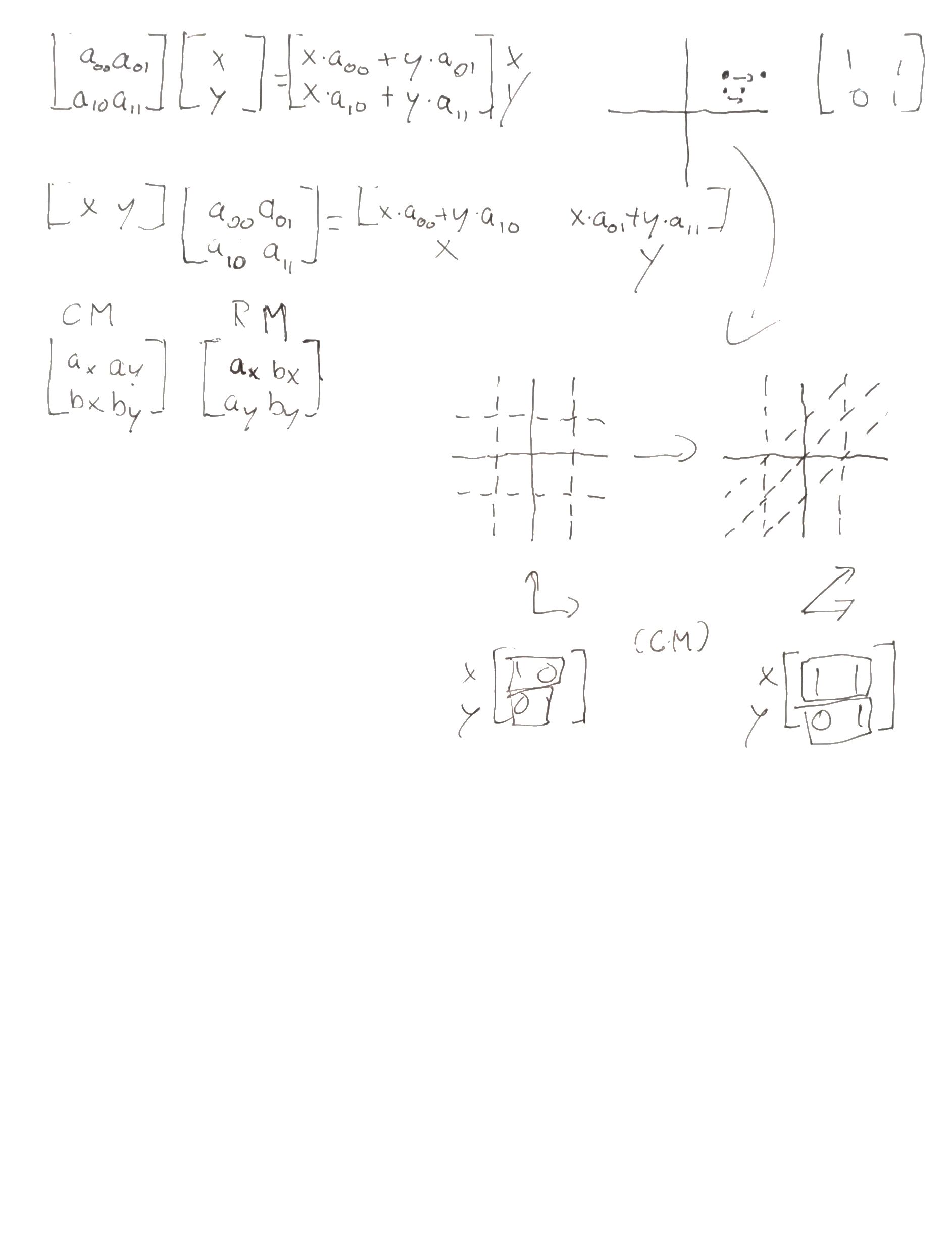
I have decided that it is not worth it to implement lookat matrices right now. I will focus my attention on implementing boxes. After I have implemented boxes, I will finish the PDP so that I don’t have to worry about that.

# Friday

## 11:45-15:15 Observe look at matrix

I still decide to implement this. The matrix isn’t a complicated one, I just don’t understand how to create translation matrices. That is a vital skill to have and I will research it.

**12:00 Research how to create transformation matrices work**

It is so obvious now. It all started making sense when I looked at points as vectors. All points are vectors from an origin. All a transformation matrix does, is transform the vectors from the orthonormal basis to other vectors. The new vector orientations is what the transformation matrix represent. This is also why rotation of 90 degrees work. You swap the vectors of the basis. 

This simple thing is exactly what the lookat vector does. We replace the old orthonormal basis with a new one.

Column and Row major transformation matrices differ from each other because the order is different. In CM (Column major) the basis vectors are represented in the rows (vector a, I now understand why people refer to it as xx,xy,xz. That is because it represents the x basis vector). Implementing the lookat matrix is as simple as inserting the new basis vectors.

**13:00**

Now that I understand the importance of order of transformation I have to refactor my camera class a bit.

I can also fix the weird rotation that I had before. By multiplying the rotation matrices independent of each other, they multiply the rotation based on the original basis. This stops the rotation from also turning the z axis.

**14:00 Finish the look at matrix**

Here is the problem. If I use one rotation matrix, the new rotation doesn’t care about the current rotation. The x axis rotates around as if the old x basis vector still existed. It doesn’t use the right basis vector.

I find this weird because it should just transform the old rotation matrix with the new one. Either way, splitting these matrices up solved the problem. One matrix keeps track of all the rotations about the x axis and the other about the y axis. This isn’t a solution though. Because of this, I can’t modify the camToWorld matrix anymore because rotations are kept from each other. This means that I can’t apply the look at matrix because it has all the rotations in one matrix. I’m a bit stuck on this.

**15:15**

I will ask this to the teachers. I should probably also research how to do camera transformations instead of figuring it out myself. I will focus my attention on implementing the cubes in the ray tracer. I have spend quite a few hours with barely any change, but I do understand how matrix transformations work now and how each element affects the final point in a scene.

## 15:15-16:00 Add some QoL changes

I’ll make the user be able to change the size of the window in the config so that you don’t have to resize it every time. I also will make the frames render partially if the fps is too low so that you can see the progression.

## 16:00-16:10 Fill out today’s work log

## 17:45-18:45 Research intersection with AABB

I need to research different ways to intersect with an axis aligned bounding box. This is because one of the ILO’s is being able to justify certain algorithms over one another.

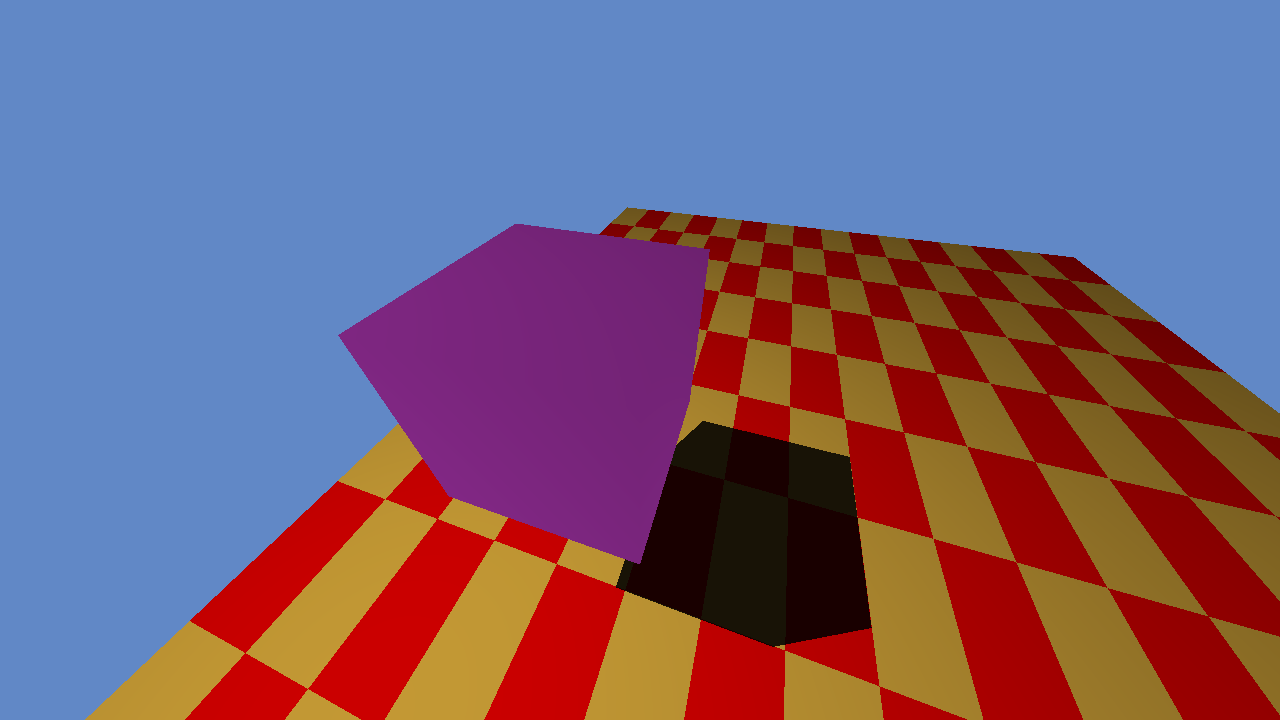
After researching for half an hour, I found but one method. The [slab](https://tavianator.com/fast-branchless-raybounding-box-intersections/) method. Everything that I have found uses the same or a slight variation of it. Because I can only find this algorithm for AABB, I will research and implement it.

The slab method works by imagining planes parallel to the box’s faces. A space will be created in between the planes. It clips the ray and if there is a part left of it then it is inside of the box.

We define a box with 2 corners, one for the min and one for the max. If the max is smaller than the min, there is no intersection.

## 19:15-22:35 Implement AABB

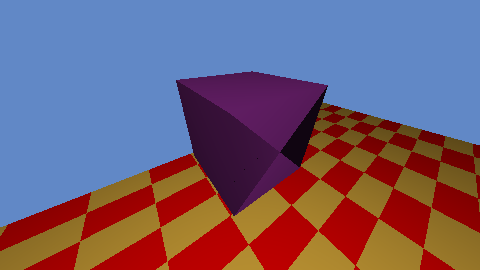
Apparently, -2 is smaller than -1. I guess that makes sense but I didn’t expect that.



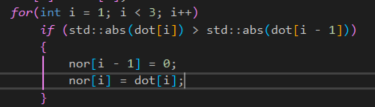
That looks nice. Now we just have to find a method on how to get the normals of this AABB.

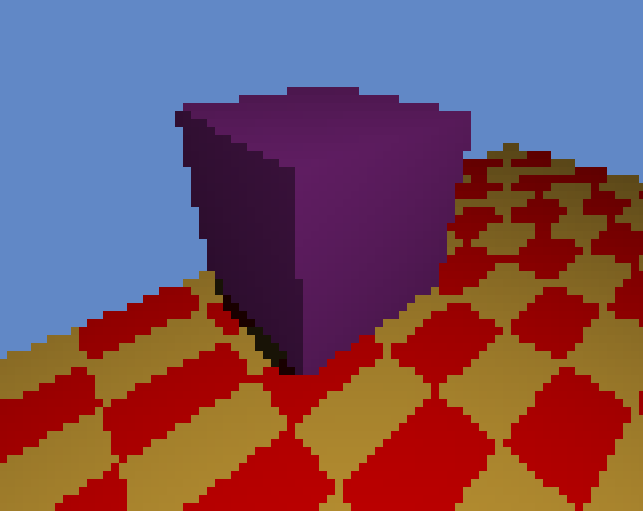
**21:00 Calculating normals**

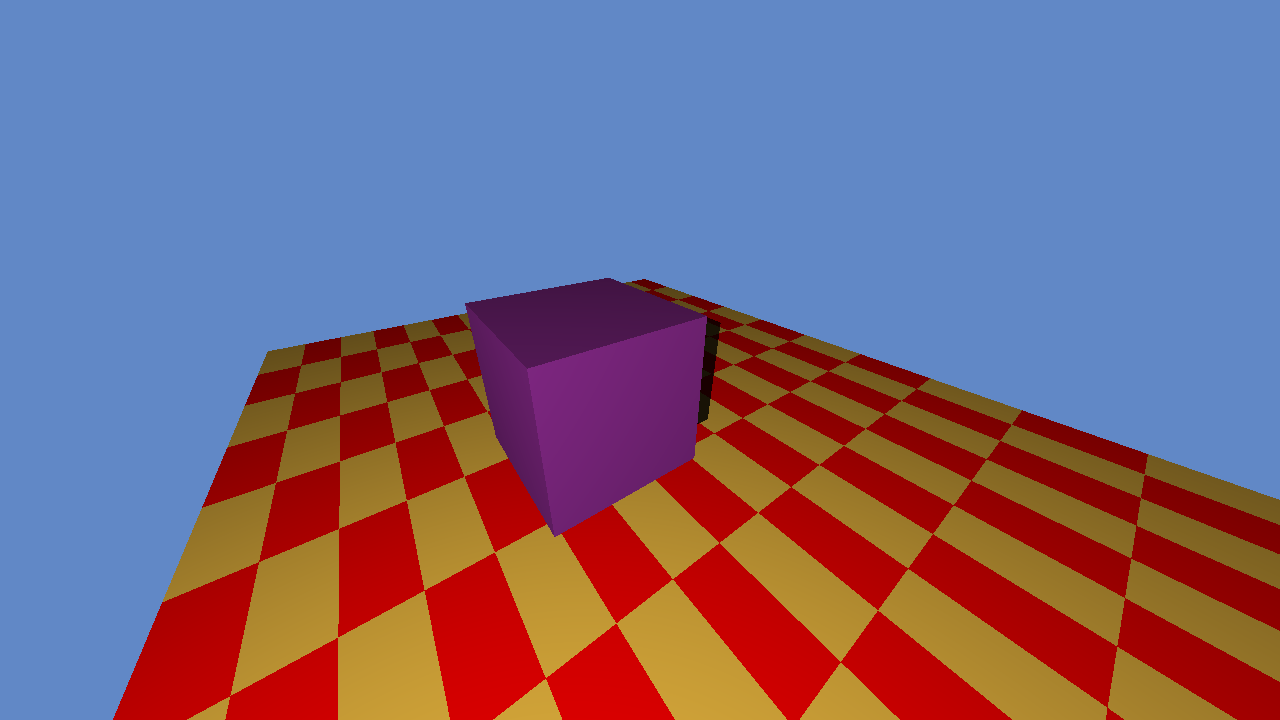
30 minutes later I still haven’t come up with a real solution. I thought that maybe I could dot all the axises with their basis and use that information to determine the direction. If the dot with the x would higher than 0.5, the intersection would be at the right side. But this doesn’t work because normalizing a point will increase it to the square root of 2 divided by 2. I could of course divide by that number but I’m not sure if that is really solving the problem. It sounds more like a hack. (and it doesn’t work as you can see). I’ll research articles online for this

**22:15**

It’s not as I expected. The shading is weird. The way I do it now is by dotting the vector to the intersection point with all the basis vectors. The largest dot product should be the correct one because the vector aims the most into that direction.

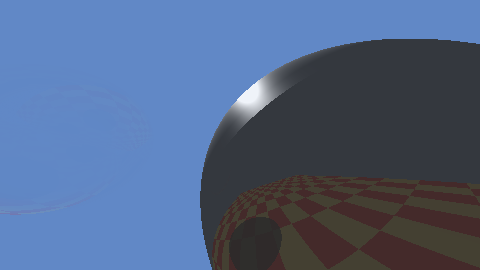
For some reason, the intersection point has a z value of -2.9. The center of the cube is located at -3 but the cube also has a depth of 2 meaning that the hitpoint should at maximum have a z value of -2.

In this loop, if y is smaller than x but y is larger than the rest, x never gets set to zero.  


Changing that bit of code fixed the program. That’s a nice cube. Let’s try it with a higher resolution.[](https://drive.google.com/open?id=1T2J58xYKcWbvWSJcdiKppOidMK3dbAnN)

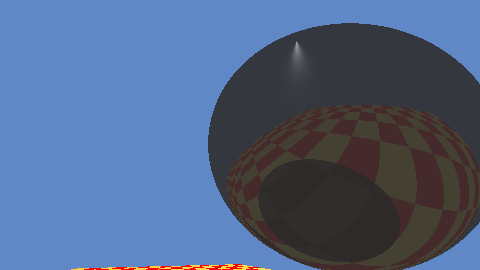
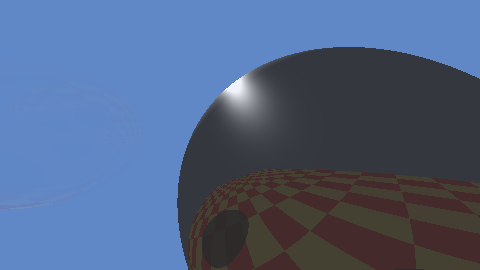
Looks great. You can click on it to expand it. I have just created a method to calculate the normal of a cube. Though it was not that complicated, more stupid mistakes on my part.

## 23:00-23:40 Fix Blinn Phong shading

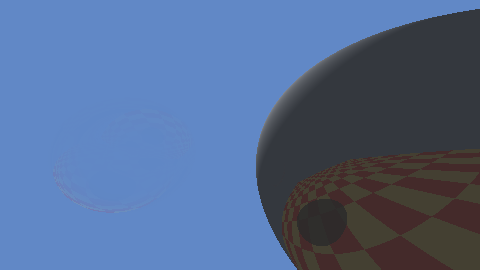


I’ve had this artifact for a while. I assumed that my angles were just weird and that is was a one time thing but the more I flew around with my ray tracer, the more I started seeing this occured. I’m pretty sure that this is not alright.

It looks like Blinn-Phong shouldn’t be affected by the shadow ray. It sounds quite weird because how can a surface have highlights if it is not exposed to a light source but this also looks incorrect.



This also doesn’t seem correct. The right picture defies all that I know about light. I’ll probably have to look at the formula again

Observing the formula I thought that I could be clever and while calculating the half vector, I just normalized it instead of dividing it with the two component vectors. The reason why you have to divide it by the length of those two vectors, is because if you normalize it, the dot product will still be very high for when you are colliding with it at a very large angle. If you add two almost perpendicular vectors with each other who are in opposite directions, and normalize that vector, it will always be the same length at a point when dotting it with the normal. This is not how Blinn-Phong shading works. Solving that gave me this result.

# Saturday

## 12:40-12:55 Fill out today’s work log

## 12:55-14:00 Implement method to allow arbitrary size AABB normal calculation

I have thought up a new method to calculate the normals of an AABB. Instead of dotting the hitpoint with the axises, we can assure that the direction to the hitpoint (that is hit - center) always has a component of 1 if it were a unit circle. We can calculate the width, height and depth of the primitive (a surface that the ray tracer can render) because we have the min and max point. I now have two methods.

I can make every AABB of unit and use matrices to transform the into world space or I can just add 3 variables that contain the width, height and depth. I’m a bit afraid that I want to overuse matrices now that I understand them because they can contain so much information and are very flexible to use, but I will discuss that with the teachers. Right now, I have no real reason to not use the clear way of defining variables although it could be nice experience.

The algorithm works as follows:

We have a size vector that contains the absolute width, height and depth divided by two. When we calculate the hitpoint, there are a total of 6 faces that it can intersect with. We know that when it intersects with a face, one of the point’s component should be of the same value of one of the size vector components. We will use a little bias because floats have precision issues, but this should work.

**13:30 Implement new way to calculate normals**

**13:50** **Finished implementation**

To show a bit of pseudocode

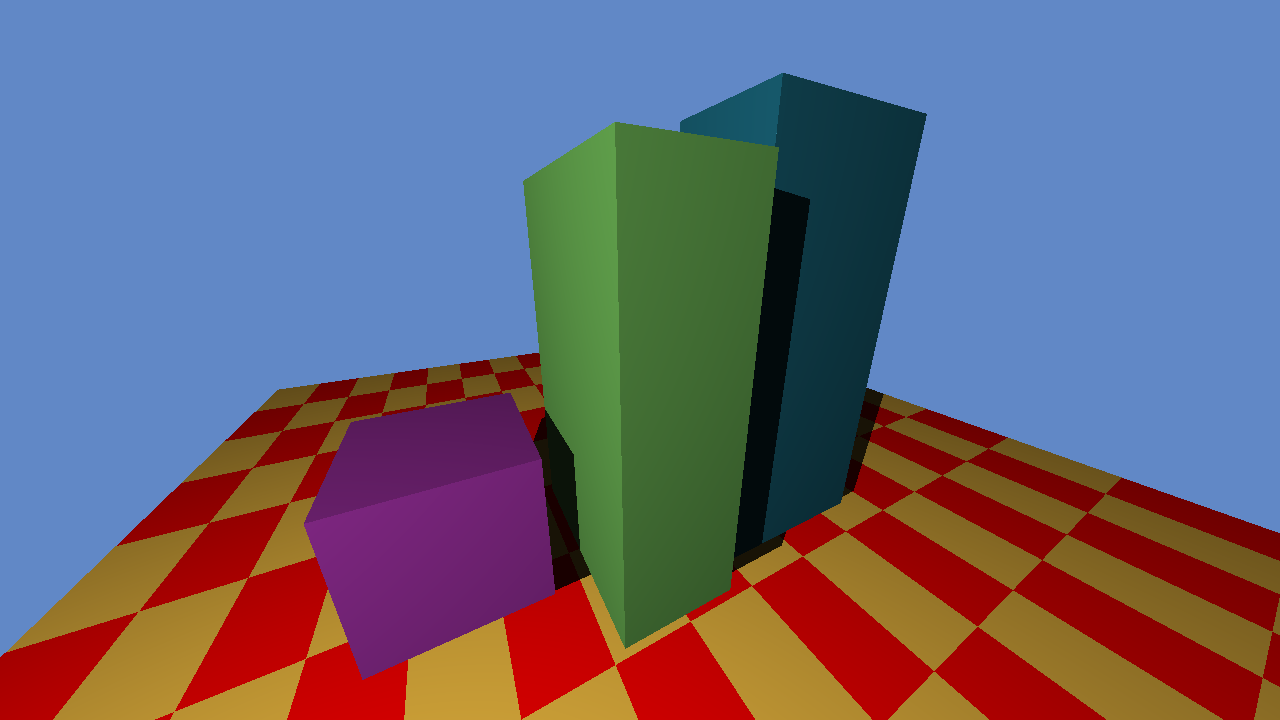
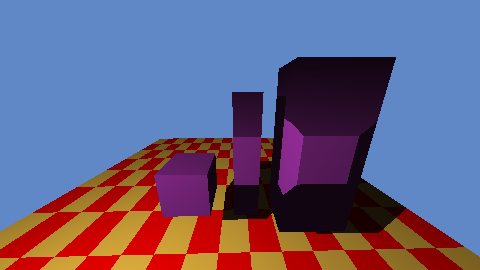
Dir = point - center

For every component

if(abs(dir) < size - bias)

Dir = 0

Return dir

That is all. It is a very simple function that relies on the fact that a hitpoint must have at least one component that is equal to the size. This is the result:

## 14:10-15:30 Implement YAML support

Fortunately, YAML (yet another markup language) has a [site](https://yaml.org/) where it lists many parsers including c++. There is only one that is made specifically for c++ which makes my choice easy. I don’t consider making my own parser right now as a option. So let’s link it.

Unlike our header JSON file, the YAML library is a bit more complicated. We have to build the library ourselves. The author uses CMake for this. To my knowledge CMake is a program that builds solutions (if we talk in visual studio terms). This allows for easy cross platform programming as everyone can just build the appropriate files for their compiler.

We will also build a solution file, but instead of using that solution, we will use the project file and add it to our solution. We are also able to create the project on our own by making a new project and adding the source files to the project, but I am curious to see how CMake works. I may have to use it in the future (not now because it is a requirement to use visual studio for this course).

After all the annoying CMake stuff, I finally got hit with it. The project contains warning because I use warning level 4. I don’t feel like changing the code of the library so I will just make a new project and add it to my solution. Then I can disable the warning levels.

I did not expect this, but apparently the warning level of the current project is used when building files. This means that making the whole project for this was redundant as well but at least I learned that. Unless there is a way to exclude projects from warning levels 4, I don’t know how to implement is without it or I have to modify the code myself.

I decided to build the solution with CMake, build it myself and add the libraries to the project. This is because I don’t want to include a project in my solution which I won’t modify.

Theres is almost no difference between the two, but it may be easier to update because I only have to update the dependencies on perforce instead of a whole project.

# Sunday

## 10:50-11:15 Fill out yesterday’s and today’s work log

## 11:25-12:15, 13:45-15:35, 18:05-19:05 Finish PDP

**11:50 Results**

**Personality type: [“The Advocate” (INFJ-T)](https://www.16personalities.com/infj-personality)**

**Individual traits: Introverted – 74%, Intuitive – 51%, Feeling – 82%, Judging – 58%, Turbulent – 51%**

**Role: Diplomat**

**Strategy: Constant Improvement**

First of all, look at intuitive and turbulent. They are so close together. I got very close to adventurer or defender, but reading the introduction of all three, advocate resonated with me the most.

**18:05 Do core quadrants**

## 20:30-21:15 Finish learning log