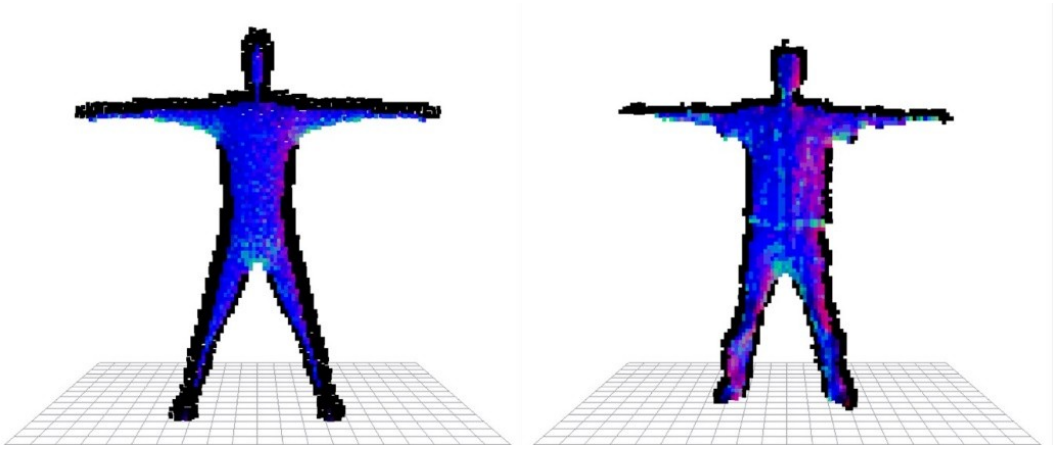
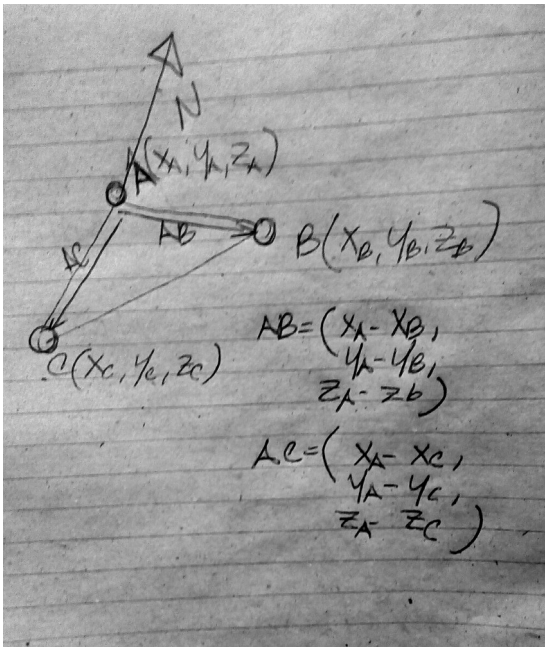
Tier 2

Question 1) How to get the facing direction of a triangle given the position of its vertices?

In the 3d space a plane is defined by 3 points as minimum. This plane has as property, a vector that is perpendicular to the whole surface of the plane (it describes the facing direction of the plane), this vector of 3 dimensions is known as the normal vector of the triangle. It has multiple applications, from render illumination to dynamic engines. In my case, I used the normals as a descriptive attribute for the generation of a pixel classification model in parts of the body from depth images obtained with the Kinect sensor, displayed of the next images.



The direction of the normal depends on the order in which the points are found. Next figure shows a triangle formed by points ABC.



The normal computation is made given the next formula.

Normal = (AB)x(AC)

where:

AB = A – B = Vector3(Xa – Xb, Ya – Yb, Za – Zb)

AC = A – C = Vector3(Xa – Xc, Ya – Yc, Za – Zc)

The normal is basically the cross product of 2 sides of the triangle.

In this case the normal is.

N = (ABy * ACz – ABz * ACy,
ABz * ACx - ABx * ACz,
ABx * ACy – ABy – ACx)

The magnitude of the normal vector almost always will be different to one, in case of needed a normalized normal you have to divide the vector by its current magnitude.

Question 2) How to calculate the area of a triangle given the positions of is vertices?

Similar to question 1, to solve this problem most be computed the cross product of two sides of the triangle. We can use the same method used on question 1, where we compute the sides of the triangle es the difference between two of its points (AB = A – B) and (AC = A – C)
The area of a triangle es the half of the area of the parallelogram that is created if you mirror one of the points of the triangle by the opposite side.

So, the area of the triangle for this case is as follow:

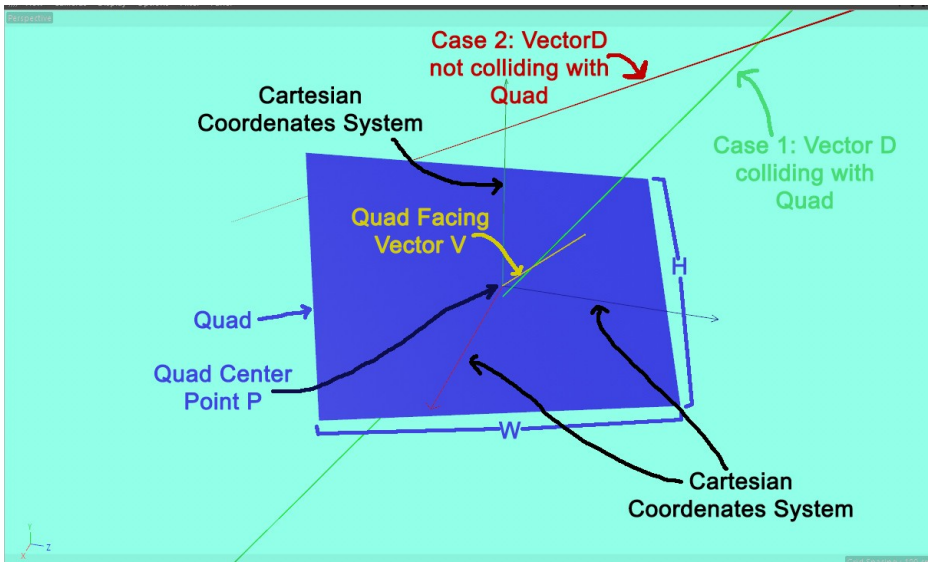
Area = |AB X AC| * 0,5

in other words the area of a triangle is the half of the magnitude of its “unnormalized” normal.

Question 3) In a VR environment, How do you know whether user is looking inside a Quad (size WxH and Position P and facing vector V)? Considering a vector D of current user facing direction.

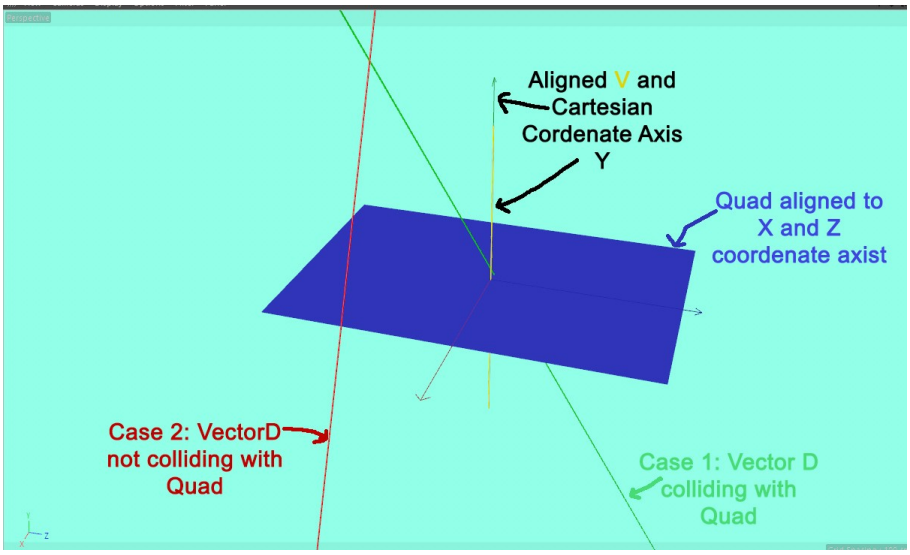
In unity 3d this problem can be solved via physics raycast (you have to raycast the vector D from the position of the player as the origin Vector), assuming that the quad is described by a collider plane.

In order to solve the problem in a mathematical fashion we propose the following scenario.

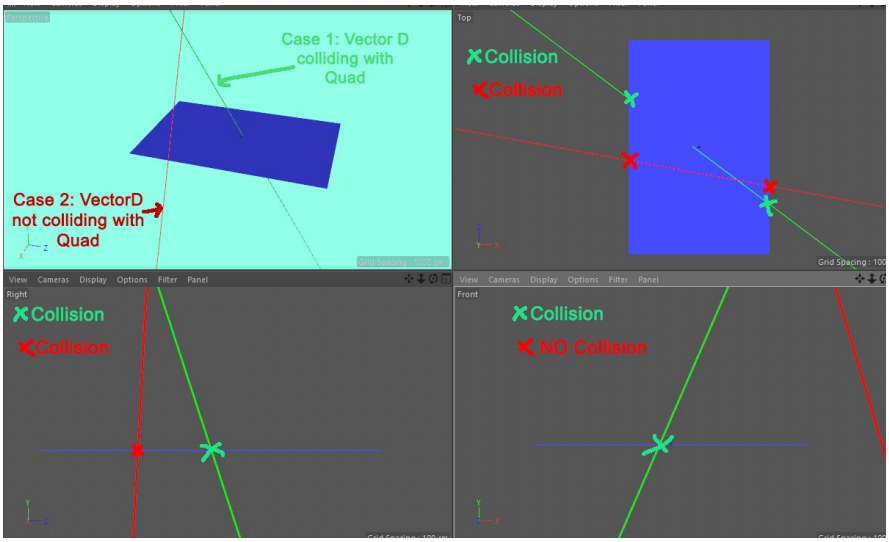


In the previous picture is represented the situation with two variables, a case where the vector D is not colliding with the quad and a case where the vector D is colliding with the quad there for the user is looking inside the quad.

The first step to solve this problem is to transform all vectors to a position where the Quad Facing vector V is aligned with one of the coordinates axis and the quad sides are aligned with the other two coordinates axis in the Cartesian system. This can be archived through a multiplication of all points by a transformation matrix that makes the necessary rotations. The result of this operation is displayed on the next image.



-Now you can project all vectors and surfaces in the 3 Cartesian planes (XY, YZ, XZ) and from this point all you have to do is check if there is a collision (in 2d) between a line and the shape that resulted from the projection, as shown the image bellow (Top left - perspective view, top right – ZX projection, lower left – YZ projection and lower right YX projection):

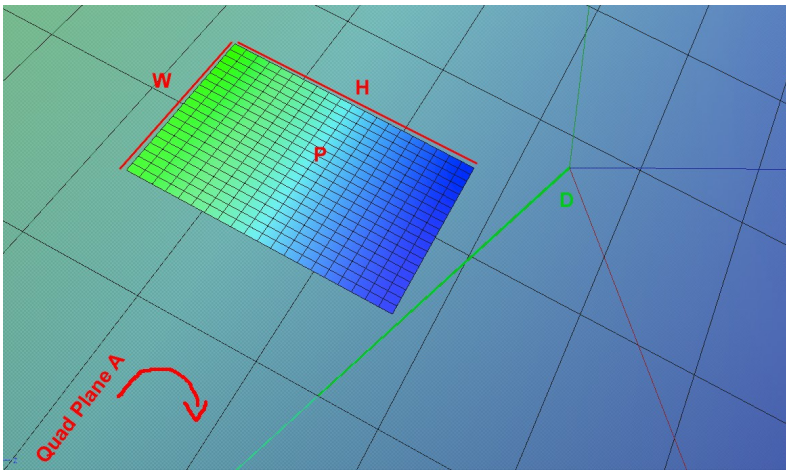


-if there is collision in all 3 planes you can say that the user is looking inside the quad. If there is not collision in at least one of the planes the user is not looking inside the quad. Both cases are shown in the next pictures.

Also there is a hard way of doing this operation. By computing the plane surface function (called Plane A) witch is the following form:

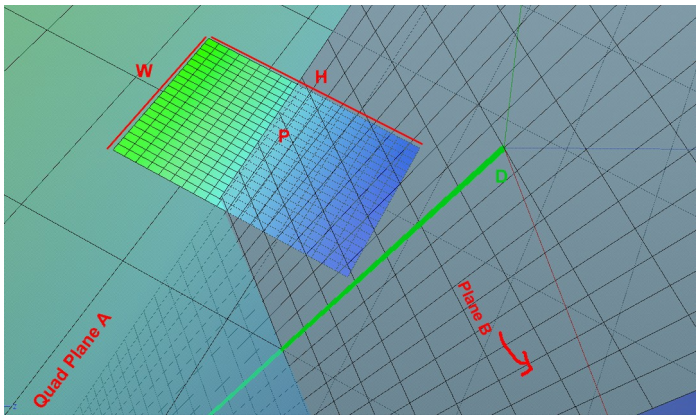
Plane A = Ax + By + Cz + D= 0

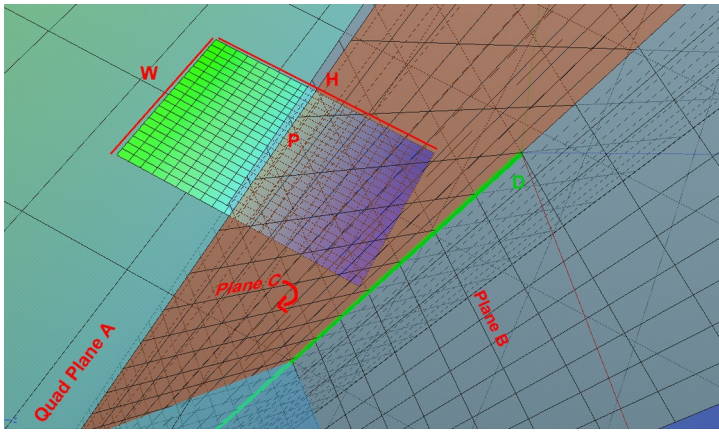
Where A, B and C are “inclinations” on its respective coordinate axis. And D is a displacement of the plane the next figures describes the situation:



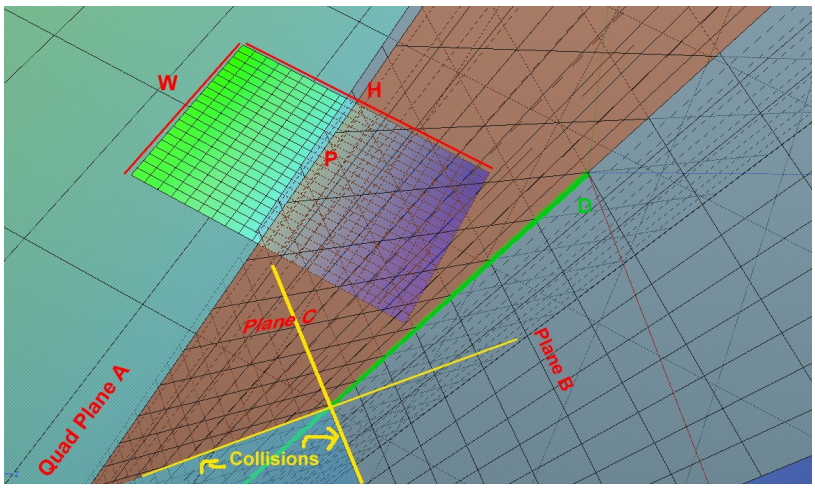
We assume that all points are relative to the point of view, this is easy done via translation.

Next step is to create two planes (through the same formula). Plane B (the first one), is made using vector D and a random vector (the origin of the facing direction and the random vector are the same (0, 0, 0)). The second plane, plane C, is made using the vector D and the normal vector of the plane A(meaning that plane B and plane C are perpendicular). The following images shows this steps.

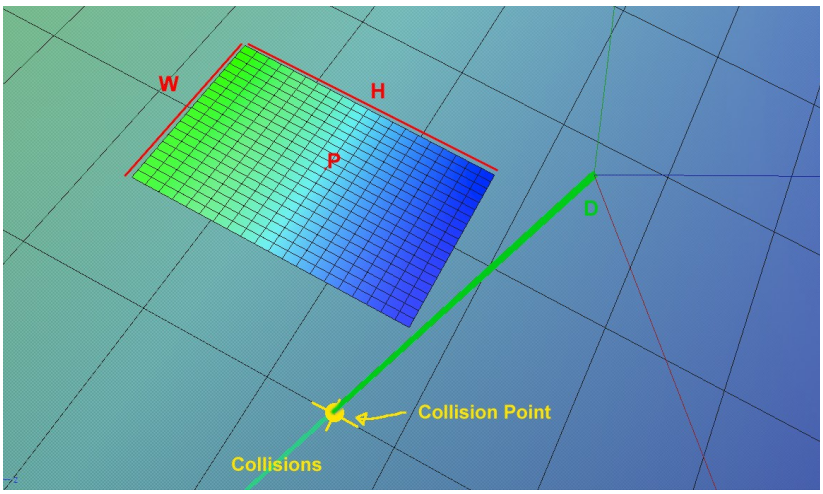


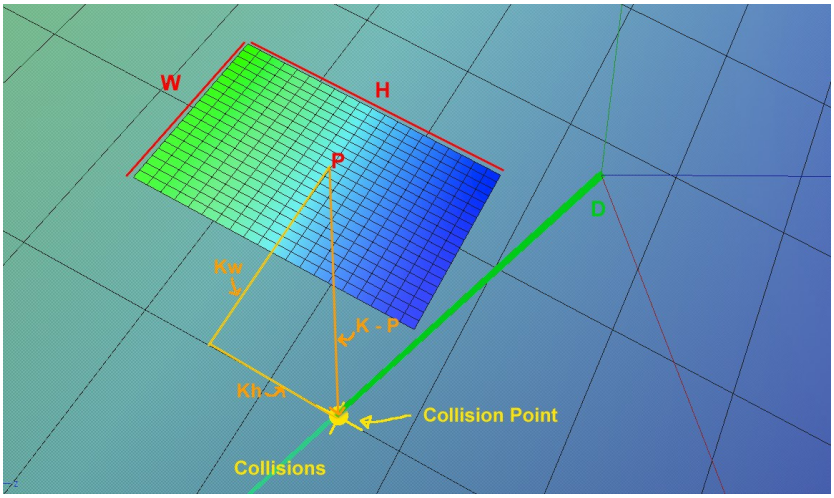


Next step is computing if there are collisions between all tree of our planes, this is done by the planes formula. If (there is a collision in PlaneA-PlaneB) and (there is collision in Plane A-PlaneB) we can say that the line collides with the Plane A. The following image shown this step



If the previous condition is meet, we can compute the collision point (K) between the three planes in a similar way. Then is only a matter of checking if the distance between K and the position P is inside the quad W x H. This is done via comparing the magnitude axis of the K – P vector with the W and H. The vector K – P must be relative to the plane A (in rotation coordinates), The following images illustrate the step.


W
H
P
D
Collision Point
Collisions


W
H
P
D
Kw
K - P
Kh
Collision Point
Collisions

Question 4) How can you guarantee that a unity cubic object with size (100, 100, 1) be represented as a square with AxB pixel size? Considering a device with resolution WxH and a virtual camera in perspective mode with D degrees of fiel of view.

In unity this can be achieved with the functionalities of the class Camera, more specifically using the WorldToScreen and ScreenToWorld function. This functions transform a 3d vector into a screen point and vice versa.

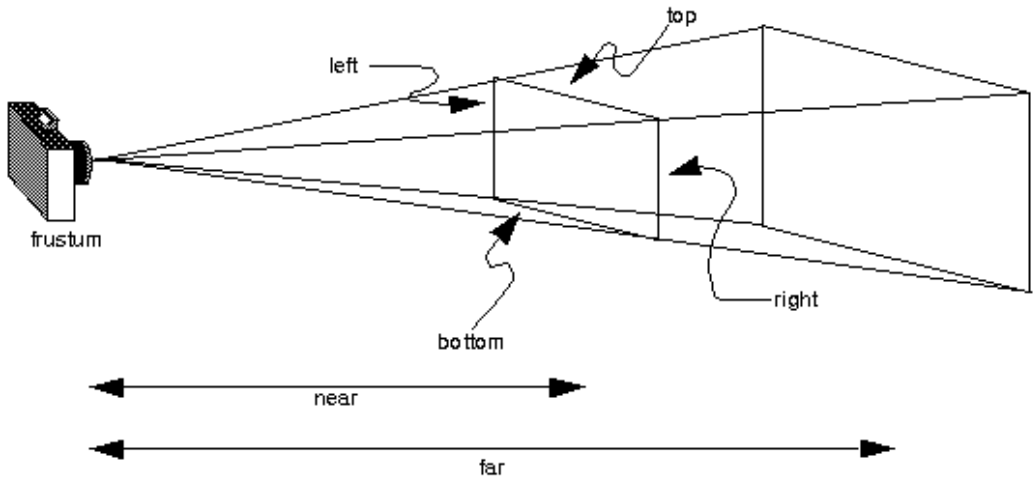
To achieve this the camera has a projection matrix witch is of the following form.

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where:

- n is near field
- f is far field
- r is the right side of the near field rect
- l is the left side of the near field rect
- t is the top side of the near field rect
- b is the button side of the near field rect

as shown in the image bellow, the near plane is the ‘projection’ plane:



the properties n, f most be defined by the user, all other variables are computed through the field of view and the Width and Height of the device.

Thanks to homogeneous coordinates (x, y, z, w) you can multiply any 3d vector by the projection matrix resulting in a camera relative point. Its also important to know the distance between the point of view and the center of the cubic geometry, we will call these measure d.
From this data one can form this vector (100, 0, d). Transform this vector to homogeneous coordinates (100, 0, d, 1) and multiply it by the perspective matrix, the result of this operation is as follow.

Simplifying the projection matrix multiplication by the homogeneous vector we obtain the formula for the width of the cubic object in screen coordinates.
That is:

$$Sw = (2n * 100/(r – l)) + (r + l) * d / (r – l)$$

As you can see the width of the object in camera projection Sw, depends on the d (distance between the object and the camera) and the width of the object itself in this case is 100. If you want the object to be of a specified size on the screen, meaning that Sw will be a number you can do it by clearing the variables d or the object width. The result of the formula will give you the amount of distance you most move the object (far from the camera) or in case of clearing the width, the size the object most have.

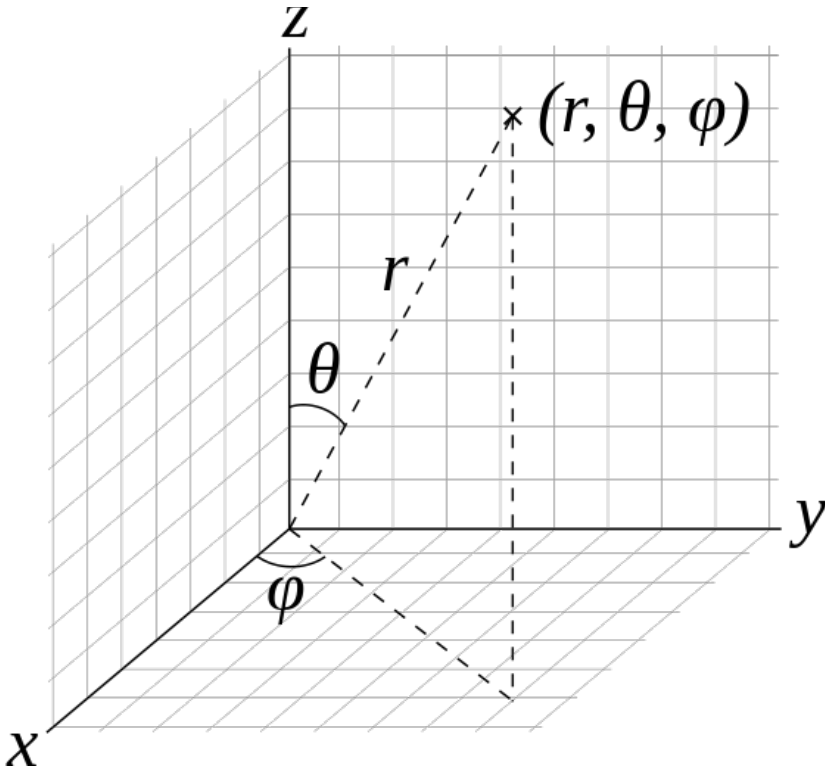
Finally is important to say that all this operations are done assuming that the object is in front of the camera, if its not, the fist step is to guarantee this condition.

Question 5) Let's create a Paint app in VR for editing 360 images. How can you know what is the current UV position user is pointing to? Considering a vector D of a current user facing direction.

A way of solving this problem is to convert the vector D to spherical coordinates, following the next formula.

$$r = \sqrt{x^2 + y^2 + z^2}$$
$$\theta = \arccos \frac{z}{\sqrt{x^2 + y^2 + z^2}} = \arccos \frac{z}{r}$$
$$\varphi = \arctan \frac{y}{x}$$

Where x, y, z are the Cartesian axis components of the vector D, r is the magnitude of the vector D and θ is the vertical inclination (-180, 180) and φ is the horizontal angle (0, 360). As shown on the image bellow.



From this point forward the UV point follows the next definition.

U = φ/360

V = (-1 * θ)/360 + 0,5

if you want to know a what pixel in the image is the vector pointing to multiply U by the image width and V by the image Height