Jacob Million, Gloria Arauja Ruiz, Andrea Uyangaa
Prof. Inbar Klang
Calculus 3: Final Project
23 March 2023

Expository Paper: Computer Graphics and 3D

Forward:

Computer graphics and 3D visualization requires complicated mathematics and algorithms in order to function. There is much that goes into creating a visual effect in 2D, let alone 3D. This paper will go into some mathematics of computer graphics and 3D visualization. We will begin with an introduction into how vectors are used for visualization before moving into a laboratory project from the calculus textbook "Early Transcendentals" by James Stewart (et al) and then exploring further research for this topic.

Introduction:

Vectors, a concept explored in calculus 3, are commonly used in computer graphics and 3D because they provide a convenient way to express geometric information. A vector has a magnitude (length) and a direction (the way in which it is going). There can also be scalers multiplied to a vector. In the context of computer graphics, we use vertices and edges to make up the surfaces of objects. As opposed to the conventional graphs many might be accustomed to, in this context we use a 3D grid with three axes: x, y and z. We can use coordinate systems with these axes to make it easier to perform calculations of objects in 3D space. We will begin to introduce this topic by exploring vectors a little more in depth, providing background which will begin to put the rest of this paper into context.

There are many good examples of how vectors are used in computer graphics: collision detection, simulating physics (for forces such as gravity you would have a vector to represent its pull), and more. Here, we will explore vectors in the context of lighting calculations (shading).

Dot Product: The dot product is represented by the formula: $A \cdot B = |A| * |B| * \cos(\theta)$. In computer graphics, we can use the dot product to measure whether two vectors point in a similar direction, and to what extent they do (or don't). This can be useful when calculating a light source on an object in order to create realistic shading which will be believable to the human eye. If the result of this calculation is positive, then the two vectors are pointing in a similar direction. If the result is negative, then these vectors are pointing in different, more opposite directions. If 0, then the vectors are exactly perpendicular (90 degrees). In real application, this is used to determine whether an object is pointed away from or towards a light source.

So how is this used in computer graphics? This would usually be done with a normal vector (calculated using the cross product) of an object and a light vector. If the dot product of these two vectors are positive, then the surface is facing the light source. From here we would calculate how much light should be reflected off the object. If the dot product is negative, then the surface is facing away from the light vector and no light would need to be reflected off the object in this case. This is a very useful tool in terms of shading 3d shapes.

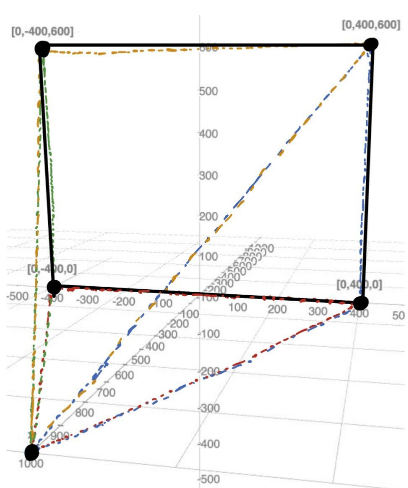
This idea of visualization in computer graphics will be explored further in the next exercise, taken from the “Early Transcendentals” textbook.

Laboratory Project, Putting 3D in Perspective:

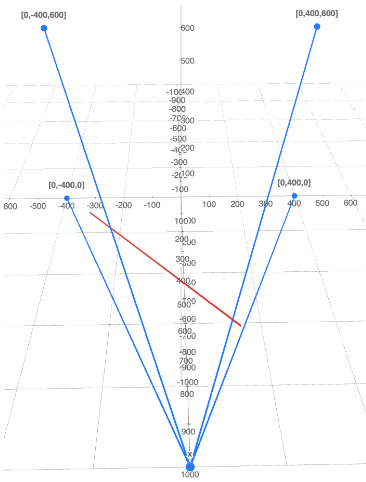
Problem Statement: In order to create the illusion of 3d space on a 2d surface (a computer screen) we must make closer objects appear bigger than objects that are further away. To do this, three dimensional objects in the computer's memory are projected onto a regular screen window from a viewpoint where the eye, or camera, is located. The viewing volume (the portion of space that will be visible) is the region contained by the four planes that pass through the viewpoint and an edge of the screen window. If an object extends beyond these 4 planes, they must be truncated before pixel data is sent to the screen. For this reason these planes are called clipping planes.

1. **Suppose the screen is represented by a rectangle in the yz-plane with vertices (0, 400) (0, -400) (0, 400, 600) (0, -400, 600), and the camera is placed at (1000, 0, 0). A line L in the scene passes through the points (230, -285, 102) and (860, 105, 264). At what points should L be clipped by the clipping planes?**

SOLUTION: Let's rewrite the coordinates of the vertices for the screen to make things more consistent.
These become: (0, 400, 0) (0, -400, 0) and we are given (0, 400, 600) (0, -400, 600).
We are also given the location of the camera, which is placed at (1000, 0, 0).



Here is the visualization of this. The point on the lower left corner represents the camera and the remaining 4 points are the vertices of the screen. From this, we can see how the clipping planes begin to form. They will essentially be the 4 triangular planes formed by connecting the top, bottom, and sides of the screen with the camera point.



We now add line L to the scene, which passes through the points (230, -285, 102) and (860, 105, 264). This line “L” is represented in red, and we want to know at what points should L be clipped by the clipping planes? From the image, it is clear that L is clipped by the upper plane, formed by the points (0, 400, 600) (0, -400, 600) and (1000, 0, 0) and the left side plane formed by the points (0, -400, 0) (0, -400, 600) and (1000, 0, 0). From here, we need to find the equation for these planes, as well as for the line, find the intersection points, and these will be the points at which L should be clipped.

-We begin to solve this problem by first finding a parametric equation for the line. First we will find a direction vector by simply subtracting the two points we are given from each other (230, -285, 102) and (860, 105, 264). => <860-230, 105-(-285), 264-102> => <630, 390, 162>. A vector equation is given in the form (a point on the line) + t<a direction vector>. Plugging our values into this formula we get: <u>(860, 105, 264) + t<630, 390, 162>.</u> Our last step is now to transform this vector equation into a parametric equation, which will be more useful in our calculations later. We simply find each value in their respective location (x,y,z). Doing this, we get the parametric equations: <u>x=860+630t y=105+390t z=264+162t</u>

-We will now find an equation for the planes that this line intersects. To do this, we will need to find the normal vector for each plane (and use a point, but this is given). In order to find the normal vector, we must first find two vectors that lie on this plane. To find a vector on a plane, we must simply subtract the coordinates of an initial point from the coordinates of a terminal point. Beginning with the upper plane, we will use (0, 400, 600) & (1000, 0, 0) for one vector, and (0, -400, 600) & (1000, 0, 0) for the other.

(1000, 0, 0) - (0, 400, 600) = <1000, -400, -600>

(1000, 0, 0) - (0, -400, 600) = <1000, 400, -600>

Now to find the normal vector of this plane we would take the cross product of these two vectors. The formula for this is: <a1, a2, a3> x <b1, b2, b3> = <a2*b3-a3*b2, a3*b1-a1*b3, a1*b2-a2*b1>. After performing this calculation, we get the vector: <480000, 0, 800000>. This vector is perpendicular to this upper plane, and can be graphed to see this if one needs further convincing of this fact. To make this a unit vector we find its magnitude with the formula sqrt(x^2 + y^2 + z^2) and divide each coordinate by the resulting number. => sqrt(480000^2 + 800000^2) which is equal to approximately 932952. Dividing each coordinate of the vector by this number we get: <.5145, 0, .8575> as our unit vector

We now can find the equation of the top plane, in the form of Ax + By + Cz + D = 0, where A,B,C are .5145, 0, .8575 respectively. All we need to find for this now is D, which can be done by plugging in any point on the plane. We will use (1000, 0, 0). This gives us: .5145(1000) + 0(0) + .8575(0) + D = 0. Simplifying we get: D = -514.5. So our equation for this upper plane is: .5145(x) + 0(y) + .8575(z) - 514.5 = 0 =>

TopPlaneEqn: .5145(x) + .8575(z) - 514.5 = 0

Completing the same steps for the left side plane we get an equation of:

<1000, 400, -600> x <1000, 400, 0> => <240000, -600000, 0>.

Unit vector of this => <.3714, -.9285, 0> => D = -371.391 =>

LeftPlaneEqn: .3714(x) -.9285(y) - 371.391 = 0

-Now we find the intersection points by substituting the values of x,y, and z in the parametric equation for the line, into the equations for the planes, solving for t, and then plugging t into the parametric equations in order to find the point on the line where we clip it.

The top plane: .5145(860+630t) + .8575(264+162t) - 514.5 = 0

=> 442.47 + 324.135(t) + 226.38 + 138.915(t) - 514.5 = 0 => 463.05(t) = -154.35 => t = -⅓

We now plug this in for t in the parametric equations: x=860+630t y=105+390t z=264+162t => x=860+630(-⅓) y=(105+390(-⅓) z=264+162(-⅓) => x=650, y=-25, z=210.

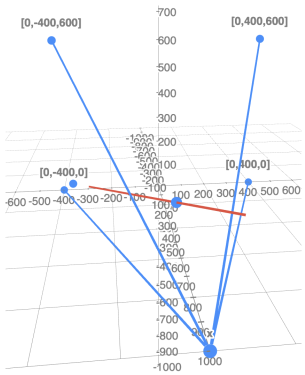
(650, -25, 210). At this point, we would clip the line (per the upper plane).

Completing the same steps for the left side plane we get:

The left plane: .3714(860+630t) - .9285(105+390t) - 371.391 = 0 => t = 1.1666x

x=860+630(-1.1666) y=105+390(-1.1666) z=264+162(-1.1666) => x=125, y=-350, z=75

(125, -350, 75). At this point, we would clip the line (per the left plane).



Looking at this visually, we can see where these two points fall, and this looks correct. (This line is continuous, although the image makes it appear as if it can not. Allow yourself to be convinced that the line intersects both of these points, and continues even further).

From here, we could mathematically confirm that this line L doesn’t intersect any of the other clipping planes, however the visual above makes clear that it does not, and we would not be able to find any intersection points with the other two planes.

2. **If the clipped line segment is projected onto the screen window, identify the resulting line segment.**

Here we just need a line connecting the camera to each end of line, and this will map to the screen.

To project this clipped line onto the screen window is relatively simple, as here we are just projecting a 2d shape onto a 2d window. Because here we are simply projecting a line from the ‘foreground’ onto the screen behind it, we can imagine 2 lines going from the camera (1000, 0, 0) towards each end of the line: one towards (650, -25, 210) and the other towards (125, -350, 75). We then can imagine these lines continue out until they each intersect with the screen at some point. We would then find these two intersection points (and would expect them to be more distant than the distance as the original truncated line as we are projecting ‘out’ here) and draw a line to connect these points on this screen. This would then be our solution to this problem.

As we already have these points, this process is fairly straightforward. We find the lines:

(1000, 0, 0) - (650, -25, 210) = <350, 25, -210> => upperLine = (1000, 0, 0) + t<350, 25, -210>

(1000, 0, 0) - (125, -350, 75) = <875, 350, -75> => lowerLine = (1000, 0, 0) + t<875, 350, -75>

We can then convert these into parametric equations to make the math easier in the next step:

<u>upperLine: x=1000+350t, y=25t, z=-210t</u>

<u>lowerLine: x=1000+875t, y=350t, z=-75t</u>

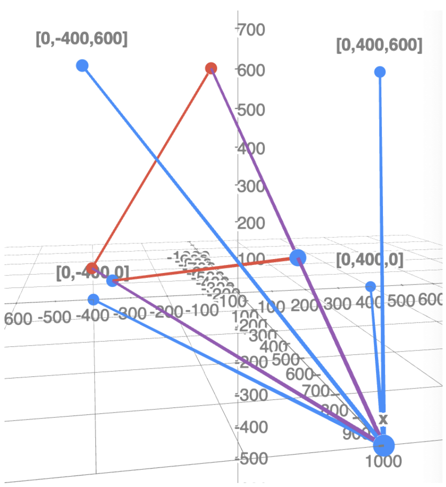
From here, we want to find the intersection of these lines with the screen, or the ‘back’ plane. This COULD be similar to the processes in step one: we will find an equation for the back plane, then plug in the values from the above parametric equations to find t, plug t back into the parametric equations and we will find the intersection points of these lines on the screen (then we will simply draw a line connecting these points as this is a simple 2d shape).

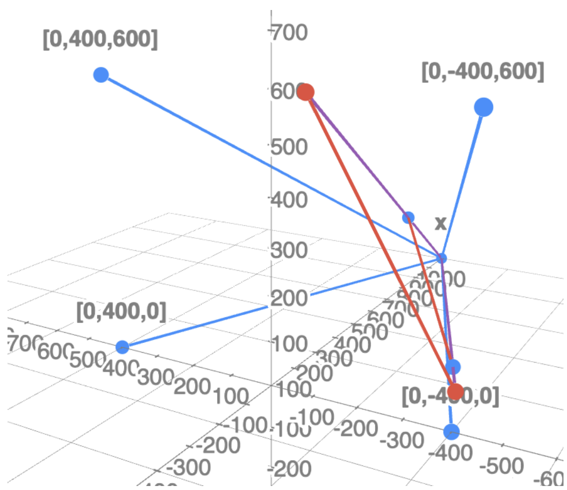
Another, easier way to go about this: The screen is at the plane x=0. So we find time t that x=0 for both of these lines, and we can find the intersection point with the plane this way by then plugging in this t value. So we find that these intersection points are:

1000+350t = 0 => t= -1000/350 = -2.857 => (0,-71.425, 600)

1000+875t = 0 => t= -1000/875 = -1.143 => (0, -400, 85.725)

Now all that we have to do is connect these two points to create the projected line segment and we will have solved this problem. In the images below, the line closest to the point (1000,0,0) represents the original (clipped) line segment. The line further from (1000, 0, 0) represents the projected line.





<u>So the projected line segment is the segment from the point (0,-71.425,600) to the point (0,-400,85.725)</u>

3. **Use parametric equations to plot the edges of the screen window, the clipped line segment, and its projection onto the screen window. Then add sight lines connecting the viewpoint to each end of the clipped segments to verify that the projection is correct.**

SOLUTION: In the two images above from problem 2, we have already completed the last step of this problem. The purple lines represent sight lines from the viewpoint (camera) to each end of the clipped segments. This lines up the way in which we would expect it to.

All that remains for this problem now is to use parametric equations to plot the edges of the screen window, the clipped line segment, and its projection onto the screen window.

To find the parametric equations, we leverage the fact that we have two points for all of the lines which we are looking for. We would find the vector equation, then convert this to parametric form.

-Screen window: here we need to find 4 equations. We will do these separately.

Top: (0, -400, 600) - (0, 400, 600) => <0, -800, 0> => (0, -400, 600) + t<0, -800, 0> =>

x=0, y=-400-800t, z=600

Bottom: (0, -400, 0) - (0, 400, 0) => <0, -800, 0> => (0, -400, 0) + t<0, -800, 0> =>

x=0, y=-400-800t, z=0

Left: (0, -400, 600) - (0, -400, 0) => <0, 0, 600> => (0, -400, 600) + t<0, 0, 600> =>

x=0, y=-400, z=600+600t

Right: (0, 400, 600) - (0, 400, 0) => <0, 0, 600> => (0, 400, 600) + t<0, 0, 600> =>

x=0, y=400, z=600+600t

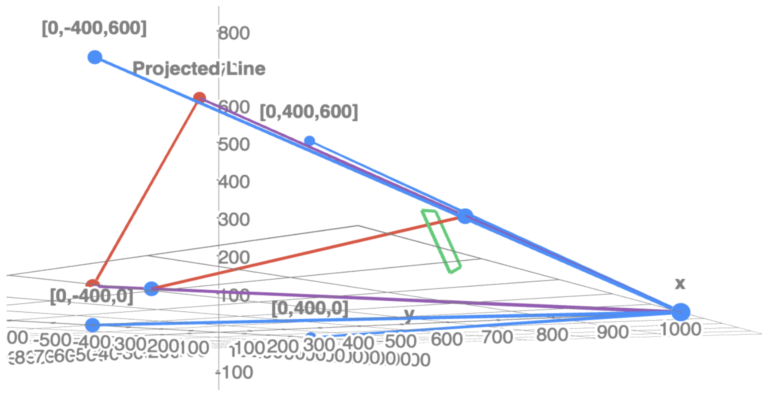
-Clipped line: (650, -25, 210) - (125, -350, 75) => <525, 325, 235> => (650, -25, 210) + t<525, 325, 235> => x=650+525t, y=-25+325t, z=210+235t

-Line projection onto screen window: (0, -400, 85.725) - (0, -71.425, 600) => <0, -328.575, -514.275> => (0, -400, 85.725) + t<0, -328.575, -514.275> =>

x=0, y=-400-328.575t, z=85.725-514.275t

4. **A rectangle with vertices (621, -147, 206) (563, 31, 242) (657, -111, 86) and (599, 67, 122) is added to the scene. The line L intersects this rectangle. To make the rectangle appear opaque, a programmer can use *hidden line rendering*, which removes portions of objects that are behind other objects. Identify the portion of L that should be removed.**

SOLUTION: First, let's add this square to our visualization to see if we need to clip any edges. From the image below, we see that there is no trimming required. We now project this square onto the screen and take care of the hidden line rendering after. To project this shape, we will simply draw a trace line from the camera, through each of the 4 corners of the shape, and find these intersections with the back plane. Another option would be to project two lines (as this would be enough to reconstruct the rectangle on the screen) but to avoid having to perform further calculations on our projected lines, we will just project all 4.



Here is a visualization of where this initial rectangle falls in the context of this problem.

We will now do the mathematics for this projection, similarly to how we did this for the line.

upperLeftCornerLine = (1000, 0, 0) - (621, -147, 206) => (1000, 0, 0) + t<379, 147, -206>

=> upperLeftCornerLine: x=1000+379t, y=147t, z=-206t

upperRightCornerLine = (1000, 0, 0) - (563, 31, 242) => (1000, 0, 0) + t<437, -31, -242>

=> upperRightCornerLine: x=1000+437t, y=-31t, z=-242t

lowerLeftCornerLine = (1000, 0, 0) - (657, -111, 86) => (1000, 0, 0) + t<343, 111, -86>

=> lowerLeftCornerLine: x=1000+343t, y=111t, z=-86t

lowerRightCornerLine = (1000, 0, 0) - (599, 67, 122) => (1000, 0, 0) + t<401, -67, -122>

=> lowerRightCornerLine: x=1000+401t, y=-67t, z=-122t

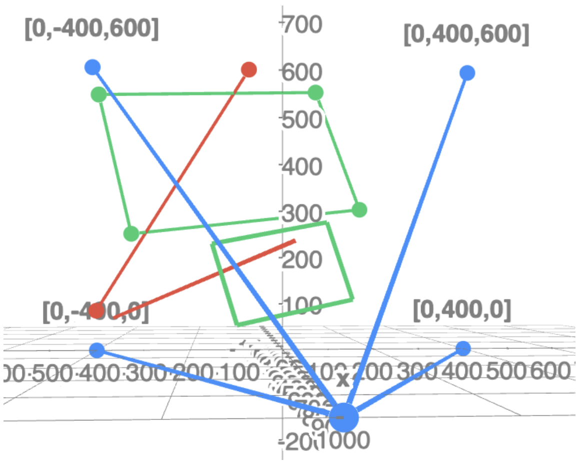
Here, we can find the t value for each of these expressions where the x coordinate is 0, plug this value back in for t, and we will get the coordinates. Then we will connect these coordinates, and we will have the projected rectangle.

upperLeftCornerLine: 1000+379t => t-2.6385 => (0, -387.86, 543.53)

upperRightCornerLine: 1000+437t => t=-2.2883 => (0, 70.94, 553.78)

lowerLeftCornerLine: 1000+343t => t=-2.9155 => (0, -323.62, 250.73)

lowerRightCornerLine: 1000=401t => t=-2.4938 => (0, 167.08, 304.24)



Here is a visualization of this projection after connecting these points. This looks correct, and if we want to assure ourselves of this, we could do trace lines here as well.

Now we just need to find the intersection points of the line and the rectangle and this will tell us the portions of the line which should be removed (from the visual, we can see that there will be two points).

First we will find the intersection of line L with the top of the projected rectangle and the line, then we will find the intersection at the bottom.

-Top line equation:

(0, -387.86, 543.53) - (0, 70.94, 553.78) = <0, -458.8, -10.25> =>

(0, -387.86, 543.53) + t<0, -458.8, -10.25> =>

<u>x=0, y=-387.86+t(-458.8), z=543.53+t(-10.25)</u>

-Bottom line equation:

(0, -323.62, 250.73) - (0, 167.08, 304.24) = <0, -490.7, -53.51> =>

(0, -323.62, 250.73)+b<0, -490.7, -53.51> =>

<u>x=0, y= -323.62+b(-490.7), z=250.73+b(-53.51)</u>

-"L" line equation:

(0,-71.425,600) - (0,-400,85.725) => <0, 328.575, 514.275> =>

(0,-71.425,600) + s<0, 328.575, 514.275> =>

<u>x=0, y=-71.425+s(328.575), z=600+s(514.275)</u>

-Solving for the intersection point with the top line we must now solve a system of equations after converting the equations into parametric form. Notice that we use different variables: "s" for the red line, "t" for the top line of the rectangle, and "b" for the bottom line of the rectangle.

Top Intersection:

x: 0 = 0

y: -387.86+t(-458.8) = -71.425+s(328.575) => t = -.689 + -.716s

z: 543.53+t(-10.25) = 600+s(514.275) But here we can plug in t in the context of s =>

z: 543.53 + (- .689 + -.716s)(-10.25) = 600+s(514.275) =>

550.69 + 7.339s = 600 + 514.275s => s = -.10

Now that we have solved for s, we can plug this into the red line "L" equation and get:

(0, -104.28, 550.16). This portion of L or "the red line" would then be hidden.

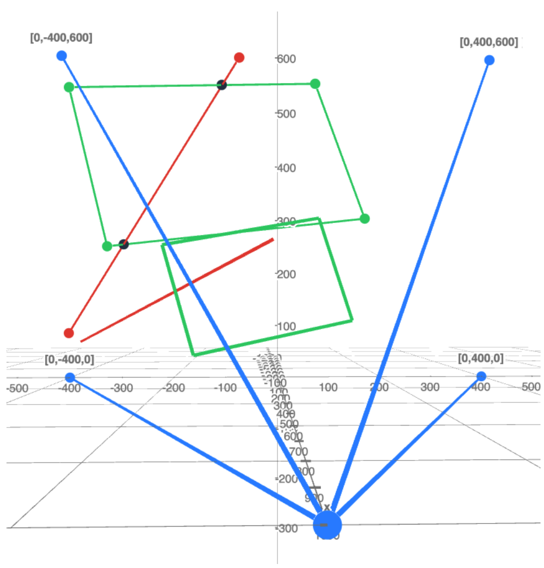
-We now solve for line L's intersection with the bottom line of the rectangle. We repeat this process with the bottom line

Bottom Intersection:

x: 0 = 0

y: -323.62+b(-490.7) = -71.425+s(328.575) => b = -.514 - .669s

z: 250.73+(-.514 - .669s)(-53.51) = 600+s(514.275) => s = -.672 => (0, -292.23, 254.41).



Now that we have both these points, we can plot them to make sure they make sense. We represent the points in which we would use hidden line rendering for the red line L with black points. Visually, this looks correct so we call this good. =>

The portions of L that would be hidden are the points: (0, -104.28, 550.16) and (0, -292.23, 254.41)

This concludes the laboratory project.

Link to the visual: https://www.math3d.org/gTlheUXN2

A few related exercises / further research on the mathematics of computer graphics:

Research Part One: Vectors and Computer Graphics

Vectors can be used in the mathematics of computer graphics, in particular, vectors are used in lighting calculations and back-face detection. Vectors can be used to solve mathematical problems that allow for an understanding of software such as "numbers systems, algebra, trigonometry, 2D and 3D geometry, vectors, equations, matrices, complex numbers, determinants, transforms, quaternions, interpolation, curves, patches and calculus" (John Vince). Types of vectors include: 2D and 3D vectors, unit vectors, position vectors, Cartesian vectors, vector magnitude, vector products, and area calculations.

a. Definitions:
    i. Scalers: Qualities of magnitude or numerical value
    ii. Vectors: Quantities that have a magnitude and a direction. They can be used to describe lines and planes in space.
    iii. Quaternions: Complex numbers in 3D form (x+xi+yj+zk) used in computer graphics for rotations about an axis
    iv. Cartesian product: a three-dimensional coordinate system

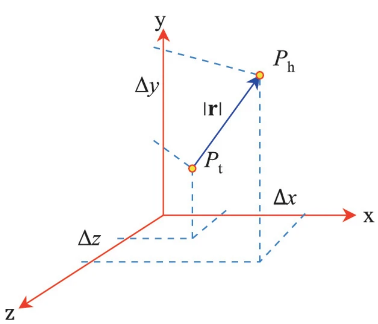


Fig 1:

A 3D image is generated by vectors such as the example to figure 1 that includes its head, tail, components and magnitude. You can also add and subtract vectors to form cumulative or non-cumulative 3D representations and graphics.

You can use a scalar product, or the dot product, to compute the projection of **r** on **s** which takes into account their relative direction/angle/position (shown figure 2). In addition, using the cross product of of two vectors, we can derive three scalar terms that form a new perpendicular vector whose " unit vector t̂*u* can now be used for illumination calculations in computer graphics, and as it has unit length, dot product calculations are simplified" (John Vince, figure 3).

Fig 2:

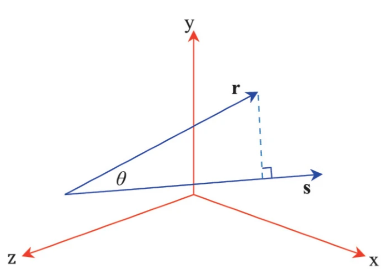
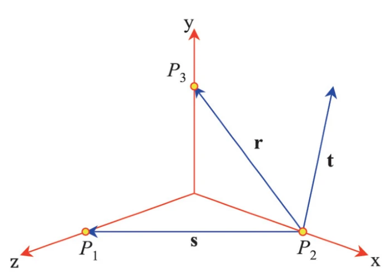


Fig 3:



2. Application Exercise Using Vectors & Distances Between Lines:

Question #1: (Ch 12, Section 12.2: Vectors, p.845, #31).
A quarterback throws a football with angle of elevation 40° and speed 60 ft/s. Find the horizontal and vertical components of the velocity vector.

Step 1: Identify given information
- The elevation angle of the velocity vector theta = 40°
- The magnitude of the velocity vector is 60 ft/s

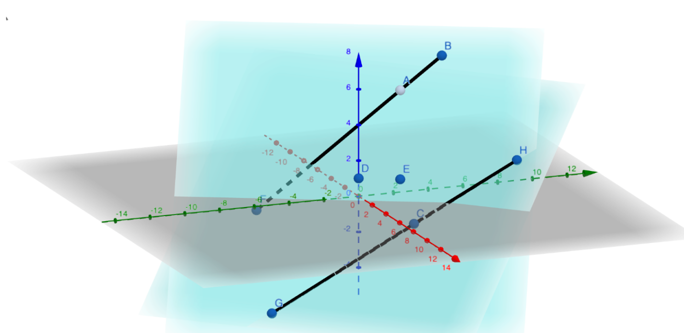
Step 2: Rewrite velocity vector to solve for horizontal and vertical components
- The velocity vector would be of the form **a** = <60cos(40°), 60sin(40°)>
  - The horizontal component is approx: 45.963
  - The vertical component is approx: 38.567

Question #2: (From Ch 12.5: Vectors and the Geometry of Space, p.874, #80).
Let L1 be the line through the points (1,2,6) and (2,4,8). Let L2 be the line of intersection of the planes P1 and P2, where P1 is the plane x-y+2z+1=0 and P2 is the plane through the points (3,2,-1), (0,0,1), and (1,2,1). Calculate the distance between L1 and L2.

- Step 1: Find the line through the points (1,2,6) and (2,4,8). First, find the direction vector for the line **AB** = <2-1,4-2,8-6>=<1,2,2>. Thus, the vector equation of line L1is **r**(t)=<1,2,6>+t<1,2,2>.
- Step 2: From P1 plane x-y+2z+1=0 and P2 plane through the points A(3,2,-1), B(0,0,1), and C(1,2,1), turn P2 into a vector equation. Use the cross product of **AB** x **AC** = <-3,-2,2>x<-2,0,2>= -4i+2j-4k = <-4,2,-4>
- Step 3: Set z=0 and add the two planes to find that x=3. Plug into either equation and find that y=4. Thus, a point of intersection is (3,4,0)
  Multiply the direction vectors of the given line and the given plane to find the direction vector of intersection: <-2,1,-2>x<1,-1,2>=<0,2,1>.
  Use this direction vector and the point to find the new line L2: **g**(t)=<3,4,0>+s<0,2,1>
- Step 4: Use the distance formula to find the distance between two lines. First, find the cross product of the direction vectors of the two planes: <0,2,1>x<1,2,2>=<2,2,-6>.
  Plug into distance formula from this point to a plane: D = l<2,2,-6>x<2,1,-2>l/((2)^2+1^2+(-2)^2)^1/2= 4+2+12/(4+1+4)^½ = 18/3 = 6 units

  Thus, the approximate distance between L1 and L2 (formed by the interaction of the two planes) is 6 units.

  Image created using geogebra.org:



Question #3: (From Ch 12.5: Vectors and the Geometry of Space, p.874, #79).
Let L1 be the line through the origin and the point (2,0,-1). Let L2 be the line through the points (1,-1,1) and (4,1,3). Find the distance between L1 and L2.

- Step 1: The equation of L1 through the origin (0,0,0) and (2,0,-1) is
  L1=(x-0)/2=(y-0)/0=(z-0)/-1
  Thus, the vector equation is **g**(t)=<0,0,0>+t<2,0,-1>
- Step 2: Find line L2 through the points A(1,-1,1) and B(4,1,3). The vector **AB** <4-1,1-(-1),3-1>=<3,2,2>. Thus, the vector equation **g**(t)=<1,-1,1>+t<3,2,2>
- Step 3: Use the cross product to find the intersection of the two lines.
  <3,2,2>x<2,0,-1>=-2i+7j-4k => <-2,7,-4>
  Finding the difference in magnitude of the two lines: <1-0,-1-0,1-0>=<1,-1,1>
- Step 4: Use the distance formula to find the distance between two lines:
  l<1,-1,1>*<-2,7,-4>l/((-2)^2+7^2+(-4)^2)^½= 13/(69)^1/2

3. Vectors in A 3D Visualization System for Hurricane Storm-Surge Flooding
https://ieeexplore.ieee.org/abstract/document/1573627
The following points explore the process of converting "high-resolution remote-sensing technology and numerical modeling," and mathematical data into visual representations of real-time landscapes, buildings, etc. in order to predict and prevent storm-surge flooding damage based on the findings in the paper "A 3D visualization system for hurricane storm-surge flooding" by K. Zhang, S.-C. Chen, P. Singh, K. Saleen, and N, Zhao.
Points:

1. 3D visualization can be used to find locations through addressed and spatial coordinates. High-resolution remote-sensing technology and numerical modeling can be used to determine the severity of damages to infrastructure as well as the water level magnitude through comparison of real-life objects.
2. Types of unit vectors can be used, alongside modeling using sums of squares due to elevation deviations (SSEDs), to detect differences in terrain such as small and consistent unit vectors used to determine small points for building roof points while large unit vectors are variable and can be used to detect vegetation points.
3. Climate change is increasing the likelihood of flooding in sensitive areas; these 3D visualization systems, modeling for animation of surge flooding, and high-resolution storm-surge modeling can be used as preemptive measures to aid the allocation of resources for people in at-risk low-lying coastal areas. This technology is particularly helpful for people that are unfamiliar with the hurricane-surge flooding and the damage that this can cause. By inputting their address/spatial coordinate, people are able to view the impact of storm surge on their specific house based on georeferenced 3D visualization data.

Research Part Two: 3D Face Recognition Based on Geometric Features:

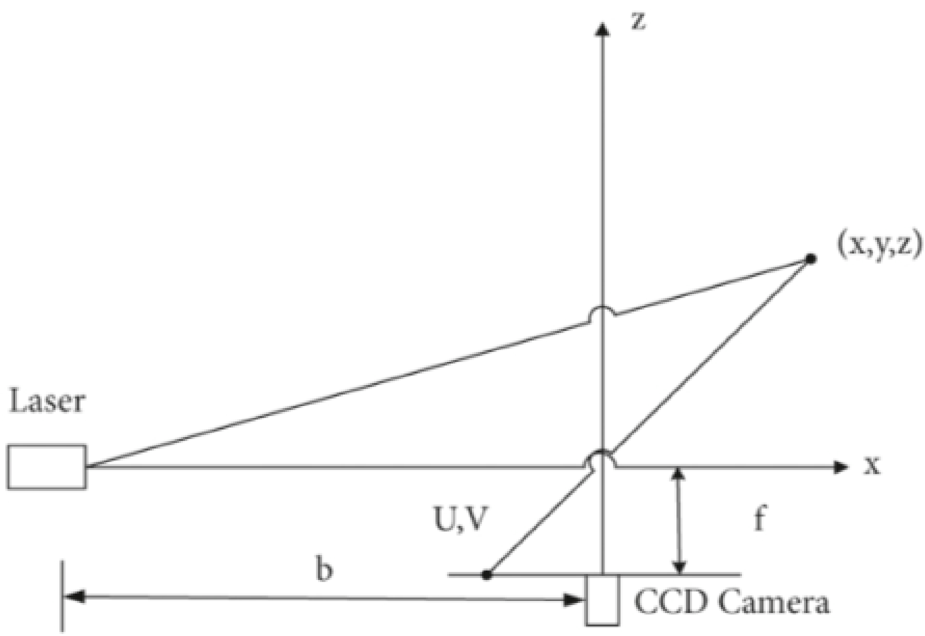
Facial recognition became a popular security choice over the years among smartphone users. This facial recognition feature has been developed and adapted by government security in China and many other countries. This technology advancement would be impossible without the three dimensional geometry and mathematics. This section of the paper will discuss 3D geometry optimization and its formula, algorithm, and graphics. The main instrument that is used in 3D face recognition is the three dimensional scanner. This scanner gained popularity not only in the medical sector, it also gained popularity for their use in technological advancement such as Computer Vision and recent popular surge of AI.

The paper "3D Face Geometry Optimization Using Artificial Intelligence and Computer Graphics" by Dan Liu explains its Calculus III use of vectors and projectors. In summary it says, to extract the three-dimensional coordinates of a human face, a high-precision three-dimensional scanner uses the triangulation technique. Through parallel, equally spaced straight lines, laser light is emitted to form an amplitude grating, which results in a linear interference fringe that maps onto the face. A charge-coupled camera can record changes in the depth and curvature of the object surface, which can alter the fringe. The laser beam of charge-coupled device and the laser beam emission angle can be used to create an internal imaging device that uses the triangle geometric relationship to determine the location coordinates or distance information of the detected point. The origin of the measuring coordinate system is the focal point of the charge-coupled device camera lens. The face's imaging point is depicted. More relevant figures and examples will be shown below.

The projection of 3D scanner:

The triangulation technique is used to retrieve the three-dimensional coordinates of the human face via the high-precision optical device known as the three-dimensional scanner [9]. It produces a linear interference fringe on the human face by emitting laser light through parallel, equidistant straight lines that form an amplitude grating. A CCD camera records the distortion of the fringe brought on by the depth and curvature of the object. The laser beam CCD and emitter angle work together to create an internal imaging device, and the triangular geometric connection may be used to determine the position coordinates or distance information of the detected point. The center of the CCD camera lens serves as the origin of the measuring coordinate system, and (x, y, z) denotes the imaging point of the face. The angle between the center of the light source and the monitored point creating a straight line and the x-axis is indicated by the letters f and B, which stand for the focal length of the camera and the distance between it and the laser projection center, respectively. Using the 3D face scanner, the face projection in Figure 1 is constructed.

Figure 1.



The system parameters in the illustration above are b, f, and based on camera calibration technology. U and V stand for the CCD camera's pixel coordinates, and the XOZ plane shows the following relationship:

$$\frac{f}{u} = \frac{z}{x},$$
$$\frac{b + x}{z} = \cos\ \theta.$$

The YOZ plane has the following relationships:

$$\frac{f}{v} = \frac{z}{y},$$
$$\frac{x}{y} = \frac{u}{v}.$$

The three-dimensional coordinates of the following measured points (x, y, z) can be obtained:

$$[x, y, z] = \frac{b}{f\ \cos\ \theta - u}[u, v, f].$$

Source Cited:

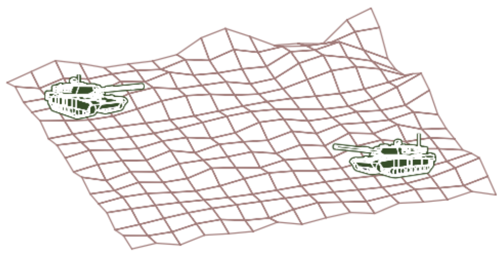
https://www.hindawi.com/journals/sp/2022/9959153/

Exercise:

81. Two tanks are participating in a battle simulation. Tank A is at point (325, 810, 561) and tank B is positioned at point (765, 675, 599).

(a) Find parametric equations for the line of sight between the tanks.

(b) If we divide the line of sight into 5 equal segments, the elevations of the terrain at the four intermediate points from tank A to tank B are 549, 566, 586, and 589. Can the tanks see each other?



Answer:

The tank A is point r_0= (325, 810, 561)

The tank B is point r_1= (765, 675, 599)

a) The parametric equation of the line between the tanks:
   - Directional vector from tank A to tank B

     (765-325, 675-810, 599-561)

     (440, -135, 38)

     Tank A is initial point

     $$\frac{x-325}{440} = \frac{y-810}{-135} = \frac{z-561}{38} = t$$

     $0 \leq t \leq 1$

Parametric Equation:

$x(t)= 325+440t$

$y(t)= 810-135t$

$z(t)=561+38t$

b)

t	z(t)=561+38t	Terrain elevation
0	561	
0.2	568	549
0.4	576	566
0.6	583	586
0.8	591	589
1.0	599	

Thus, the two tanks are not aligned and won't be able to see each other

List of Sources Referenced on the Mathematics of Computer Graphics:

A 3D Visualization System for Hurricane Storm-Surge Flooding, ieeexplore.ieee.org/abstract/document/1573627/. Accessed 9 May 2023.

"Dot and Cross Product | 3d Graphics Overview." *YouTube*, 21 Jan. 2021, www.youtube.com/watch?v=M5U4_l0Aoxc&t=3s.

"Holoportation." *HoloForge*, 5 Apr. 2022, www.holoforge.io/en/project/holoportation-2/.

Introduction to Computer Graphics, Section 3.5 -- Some Linear Algebra, math.hws.edu/graphicsbook/c3/s5.html. Accessed 8 May 2023.

Link-Springer-Com.Ezproxy.Cul.Columbia.Edu, link-springer-com.ezproxy.cul.columbia.edu/content/pdf/10.1007/978-90-481-3524-0.pdf. Accessed 9 May 2023.

Liu, Dan. "3D Face Geometry Optimization Using Artificial Intelligence and Computer Graphics." *Scientific Programming*, 22 Mar. 2022, www.hindawi.com/journals/sp/2022/9959153/.

"Neural Fields." *Computational Imaging*, www.computationalimaging.org/. Accessed 8 May 2023.

Orcos, Lara, et al. "3D Visualization through the Hologram for the Learning of Area and Volume Concepts." *MDPI*, 9 Mar. 2019, www.mdpi.com/2227-7390/7/3/247.

Stewart, James, et al. *Calculus: Early Transcendentals*. Langara College, 2022.