Math of Perspective Drawing F19

Julia Tarinelli

Final Paper

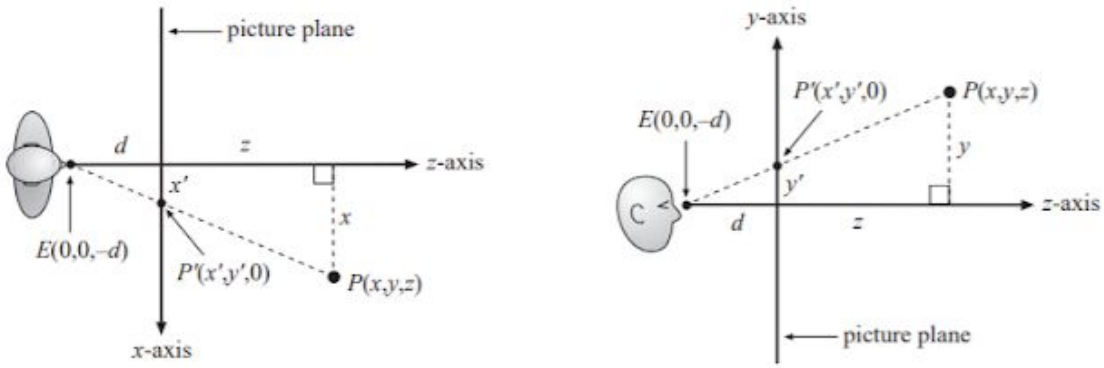
How can the illusion of a 3D object be created on a 2D surface? Both perspective drawing and computer graphics provide surprisingly similar answers to this question. Despite originating hundreds of years apart, the fields share many of the same foundations. Take the grid on the picture plane we have been looking at all semester and turn it into a grid of pixels on a screen. The exact same concepts and equations from perspective drawing apply with only a few alterations. In my project, I utilize this fact to build a computer program that can do the calculations to draw boxes in one point and two point perspective from any point of view.

The history of computer graphics goes almost as far back as computers themselves. It was only about ten years after the invention of the modern digital electronic computer in the 1940s that the first computer graphic was created in 1950. However, computer graphics remained an expensive field limited to researchers until the 1970s, when companies began developing the specialized hardware necessary for simple 2D graphics on a commercial scale. By the 1990s technology was advanced enough that personal computers and video game consoles could use 3D graphics. Further improvements in the processing speed and power of hardware has brought us to today, where computer graphics are everywhere. Not only are they everywhere on our cellphones and laptops, they are also necessary to animate movies, tv shows, and video games as well as to design all kinds of objects in CAD programs.

My project is simplified from what might be taught in a computer graphics class about linear perspective. While I am giving instructions to my program about where to draw various

shapes and lines, I am still relying on the programming language I am using to actually display those shapes on the screen. In a more advanced project I might build a renderer, which determines pixel by pixel what parts of the screen what need to be which color in order display a certain image on the screen. Computer graphics also makes heavy use of linear algebra, which is not a math class I have taken yet. While the underlying calculations are all the same as the ones we have learned, linear algebra makes them more manageable. Then, the program can execute more calculations at once as well as more complex equations in order to create more complicated, realistic, or interesting images.

Diagram from Viewpoints, page 16



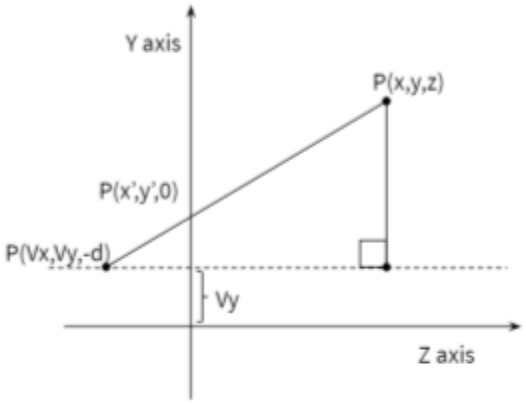
$$x' = \frac{dx}{z+d} \quad \text{and} \quad y' = \frac{dy}{z+d},$$

My program implements the perspective formula we derived in class. As seen in the above top and side view diagrams, it comes from the two similar right triangles formed by the viewer, the point in the real world, the z axis, and the picture plane. With these triangles and some algebra, we can calculate the above formulas for the x and y coordinates of the image of

the point on the picture plane. However, this formula only works for a viewer at (0,0,-d), and I want to be able to change the viewer's x and y coordinates as well.

$$x' = Vx + \left(-d \cdot \frac{x - Vx}{z - d}\right)$$

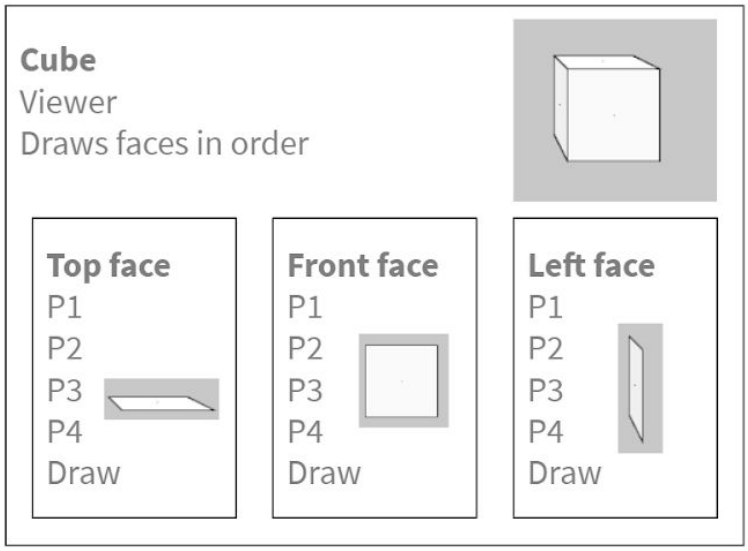
$$y' = Vy + \left(-d \cdot \frac{y - Vy}{z - d}\right)$$



Luckily, we can find the location of points on the picture plane with the viewer at any location using only a few modifications to the original perspective formula. Look at the new side view diagram above. Here, the viewer is at P(Vx,Vy,-d) and the point in the real world is at P(x,y,z). If we subtract Vy from the y coordinates of both the viewer and the point in the real world, the viewer will be on the z axis and we have the same similar right triangles seen in the previous top and side view diagrams. So, we can use the same formula from before by plugging in y - Vy for y. But, we still have to add Vy back into the equation, or our result will be as if the viewer was actually on the z axis. Essentially we are moving both the viewer and the point in the real world down by Vy, doing the calculations with the original perspective formula, and then moving everything back to its original position by adding Vy. With similar logic for the x coordinate, this is where the new formulas for the coordinates of a point on the picture plane seen above come from.

I chose to use Processing to build my program in, mostly because it is a language I am comfortable working with and it is easy to create graphics and animation with. The coordinate

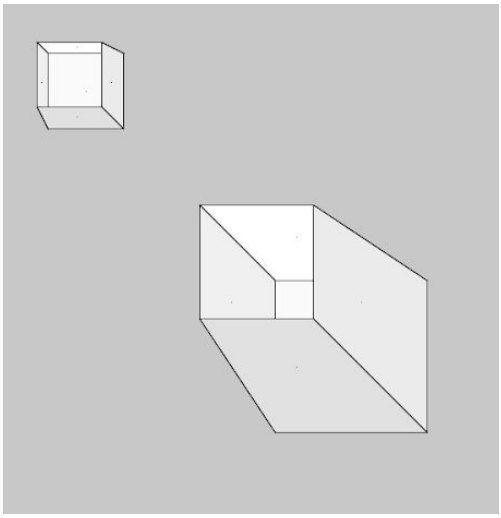
system is a little different than what is typically used in math, as the origin is in the upper left corner of the screen and the units are pixels. For example, to draw a circle 100 pixels from the left edge of the screen, 100 pixels from the top edge, and with a radius of 10 pixels all I need to do is type *ellipse(100,100,10)*. I can also make a quadrilateral with corners at any 4 points on the screen, which is what I used to draw the faces of my box.



The first step in building my program was organizing all the pieces and how they will interact with each other. I used something similar to the above diagram to plan before I started coding, where each box represents an object. While this diagram only shows the 3 faces visible at the particular angle the cube is shown at, the cubes in my program actually have 5 faces. I did not include the back face since it is never visible.

The first things I built were the faces, as these are the objects which actually handle the perspective calculations and the drawing of quadrilaterals on the screen. Each face holds 4 points, one for each corner. When it is passed the location of the viewer it calculates where each

point would appear on the screen/picture plane from that location. Then, it draws a quadrilateral with corners at those points, which is one face of a box in perspective.



Once I could draw one face in correct perspective, I built the cube object. It holds 5 face objects, and handles the order they are drawn in. To the left is an example of what happens when the faces are drawn in the wrong order. The correct order is determined by something called the painter's algorithm. Just like the background of a painting is laid down before the foreground, this algorithm states that things that are further away get drawn before things that are closer to the viewer. All I had to do to implement the algorithm was calculate the distance between the center of each face and the viewer, and sort the faces based on this value.

Now I can draw a cube at any location, viewed from any location, but only by typing the coordinates of each point into the program. I want anyone to be able to move the viewer and immediately see the results without having to go edit and run the code themselves. I thought that a slider on screen for each coordinate would be the most intuitive way to control the viewer's position. Instead of building the slider myself I used a graphic user interface (GUI) library called controlP5. I also used this library for the buttons that switch between one and two point perspective. While I could have built both myself, I thought it would take too much time for something that is not integral to the main purpose of the project.

The end result of my project is a program which displays cubes in one or two point perspective. The user can use sliders to move the viewer's position and see how the appearance of the cubes change. I hope that by using my program and being able to immediately see the results of changing a parameter, it will be possible to get a more intuitive understanding of perspective and how it works beyond just deriving the formula or drawing cubes by hand. Adding more options, like the ability to view the vanishing points and how the lines of objects extend to them could help to further this understanding of perspective if I were to work on more on this project in the future.

I also hope that my program can be an example of how computer graphics and computers in general can combine math, art, and technology in fun and interesting ways. A programmer can make art that the inventors of linear perspective during the Renaissance never could have dreamed of.

Bibliography

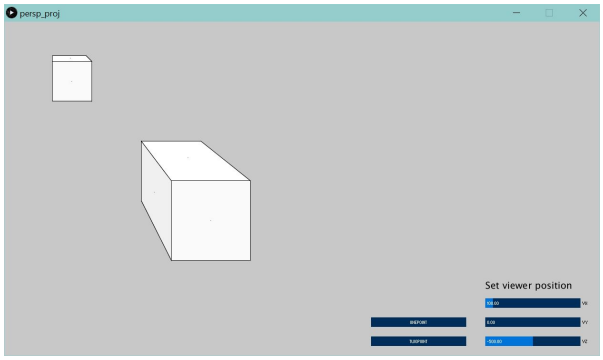
Applied Geometry for Computer Graphics and CAD, Duncan Marsh

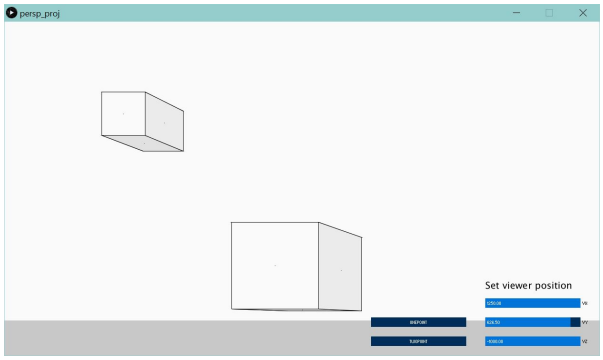
The Computer Graphics Manual, David Saloman

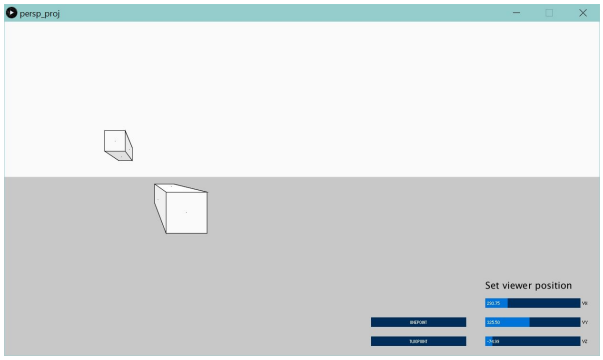
Viewpoints: Mathematical Perspective and Fractal Geometry in Art, Annalisa Crannel and Marc Frantz

Appendix: Program Screenshots

One point perspective







Two point perspective