Projection

In 1984, a future computer scientist saw a magazine advertisement for Microsoft Flight Simulator (Fig 1a) and begged his parents to buy it for him. Imagine his surprise when, after waiting for three weeks for the diskettes to arrive via mail order, all the while imagining the advanced virtual reality experience that awaited him, he started the program and saw the screen in Fig. 1b. (Forty years later, he's still not sure why the runway was purple and the sky was orange.)

Let's recreate the exhilarating disappointment of that moment. We've prepared a file that contains a simple model of an airport, with a runway and control tower, encoded as a list of 3D points. By projecting this set of points into a 2d image plane, you can create a view of what the 3D scene would look like from any arbitrary viewpoint. For example, Fig. 2a shows a view from a plane that is approaching the airport, while Fig 2b shows a view from a plane that is waiting to take off on the runway (similar to the view in Fig 1b).

Your goal in this lab is to create a program that simulates the experience of taking off from this airport. To do this, you'll need to create a sequence of images formed by projecting the 3D scene into the 2D imaging plane of a camera, as this camera moves down the runway and increases its altitude. Recall that projecting a 3D scene onto a 2d camera plane involves encoding each 3D point as a 4D homogeneous coordinate, multiplying by a 4x3 projection matrix Π, and then converting the point from homogeneous coordinates to a 2d point. The projection matrix Π can be formed by a matrix product,

$$\Pi = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \end{bmatrix}$$

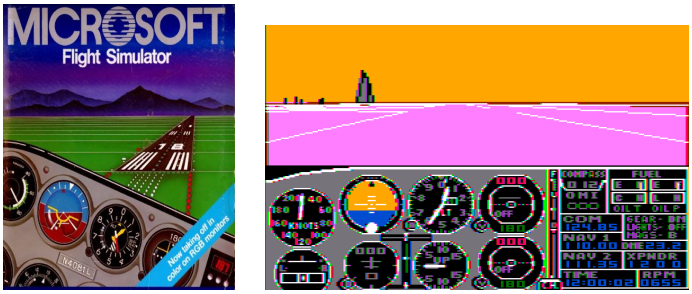
where f is the focal length of the camera, α, β, and γ are the tilt, twist, and yaw angles[1] of the camera, respectively, and (t_x, t_y, t_z) is the location of the camera in 3D space.

We've included some sample code that loads in the airport points and does a very simple animation of "flying over" the airport. It uses a simplistic projection technique called orthographic projection, which only works if the scene is very far away. Your job is to replace the animation with a new animation that starts on the ground (at an altitude of, say, 5 meters, since the cockpit window is at the top of the aircraft) at the end of the runway, and "takes off" by advancing down the runway and increasing the altitude.

Here are some hints:

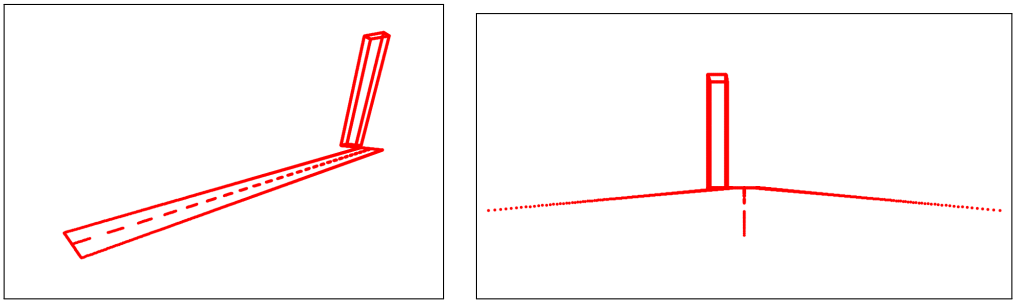
- The 3D points use a coordinate system (x, y, z) such that z is altitude off the ground, y corresponds to east-west, and x corresponds to north-south.
- The end of the runway is positioned at point $(0, 0, 0)$. The runway is aligned north-south.
- A focal length of about 0.002m should work well.
- Don't worry about creating realistic flight. Just gradually advance down the runway and increase altitude. (But for a small amount of bonus credit, have the plane fly an oval formation: take off, bank right, fly a bit, bank right, fly parallel to the runway, bank right to align with the runway, and then bank right and land.)
- In real life, the world behind the camera does not appear on the image plane because the back of the camera is opaque. But nothing in our mathematical projection model prevents points behind the camera from appearing in the image. To fix this problem, examine the sign of the third homogeneous coordinate in the projected point (i.e. the third coordinate obtained after multiplying Π and the 4-D

[1] To illustrate these three rotation dimensions, image you have a camera mounted on your head. If you nod your head, you're changing the tilt angle. If you rotate your head so that your ear touches your shoulder, you're changing the twist angle. If you rotate your head left to right, you're changing the yaw angle.



(a) (b)

Figure 1: Microsoft Flight Simulator v.1's (a) promotional material compared to (b) an actual screenshot (supposedly of the view out of the cockpit of a plane awaiting takeoff at Meigs Field in Chicago; the purple thing is the runway your plane is resting on, the white lines are the runway markings, the orange blob is the sky, and the pointy shape in the upper left is the John Hancock Center skyscraper).



(a) (b)

Figure 2: Two views of our airport, from (a) an oblique angle from above, and (b) from one end of the runway. (The rectangular structure is the airport control tower, of course.)

homogeneous scene point): points with a negative third coordinate are behind the camera and should be ignored.

What to turn in

Please submit a zip file containing your source code along with the mp4 video of your program's "flight." Please make sure that your code runs on the university's linux server prior to submission. If you used any external libraries other than Pillow, Numpy, or SciPy, please include a README file explaining why you did so.

When you submit your zip file, please put everything in a folder named `<username>_lab4` and zip that file.

Again, if you have any questions, please ask us on Piazza!