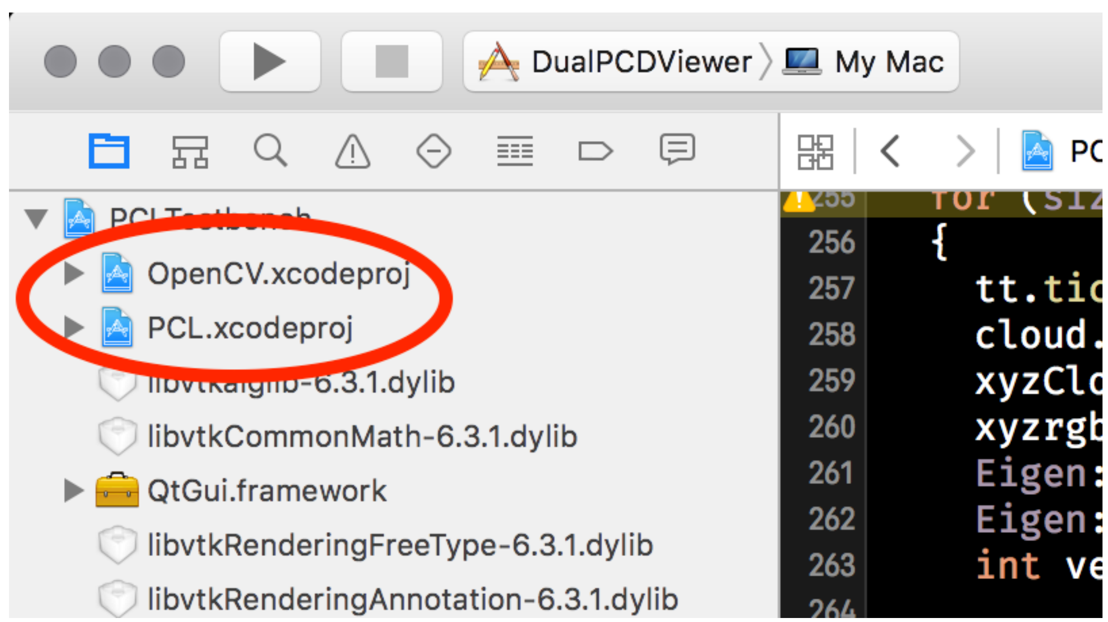
Matt’s updates for Feb 25th

Part 1: Combining OpenCV with PCL



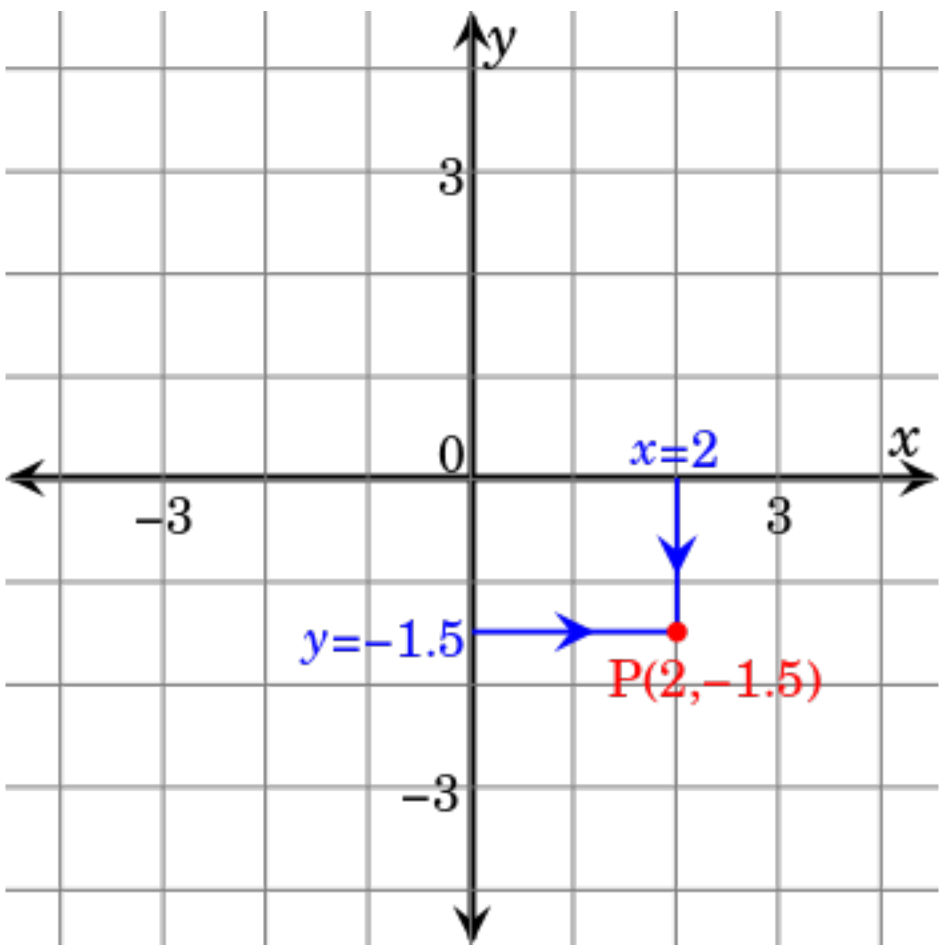
Code sample

```
cloud.reset(new PCLPointCloud2);
xyzrgbCloud.reset(new PointCloud<PointXYZRGB>);

if (pcdReader.read (filepath.c_str(), *cloud, origin, orientation, version) < 0)
  std::cout << "problems opening file" << std::endl;
fromPCLPointCloud2(*cloud, *xyzrgbCloud);

if (xyzrgbCloud->isOrganized()) {
  src = cv::Mat(cloud->height, cloud->width, CV_8UC3);

  if (!xyzrgbCloud->empty()) {

    for (int h=0; h<src.rows; h++) {
      for (int w=0; w<src.cols; w++) {
        pcl::PointXYZRGB point = xyzrgbCloud->at(w, h);

        Eigen::Vector3i rgb = point.getRGBVector3i();

        src.at<cv::Vec3b>(h,w)[0] = rgb[2];
        src.at<cv::Vec3b>(h,w)[1] = rgb[1];
        src.at<cv::Vec3b>(h,w)[2] = rgb[0];
      }
    }

  }
}
```

Demo

Part 2: Homogeneous Coordinates

Background: Cartesian planar geometry



Familiar 2-d: Cartesian Coordinates

- points are tuples (x,y)
- transforms are 2x2 matrices that scale and rotate the plane
- i.e., left-multiplying a vector by a matrix results in a scaled and rotated vector:

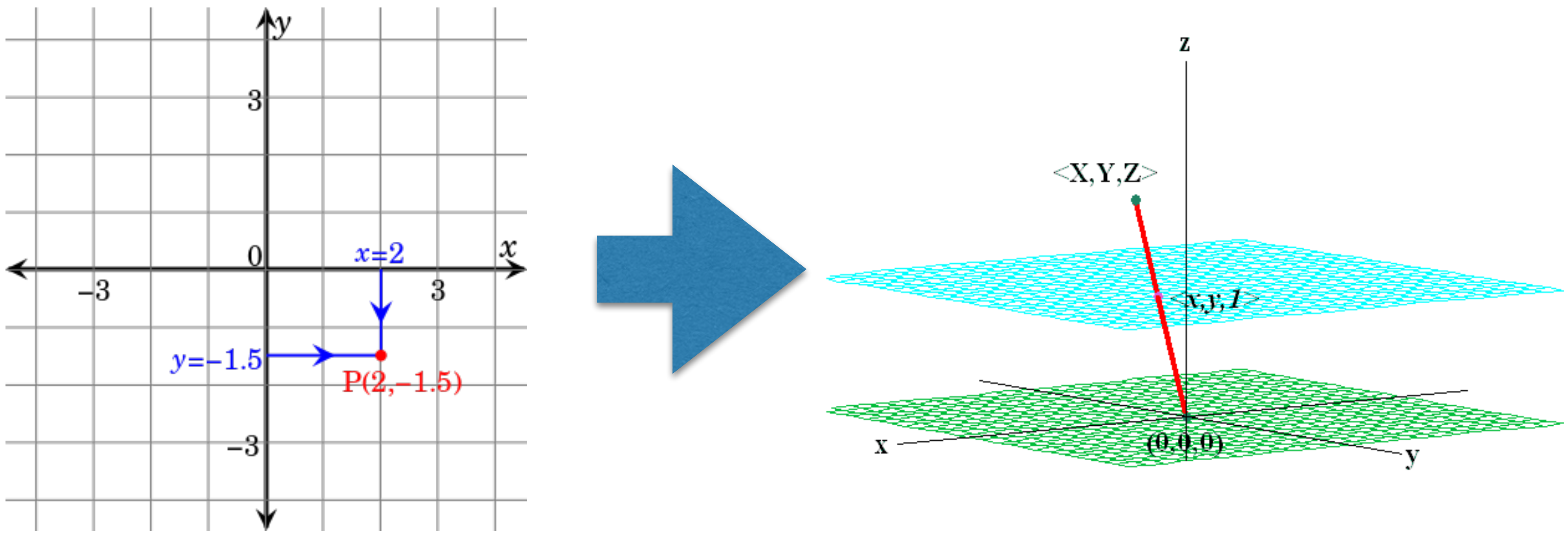
$$Av = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times (x, y) = (ax + by, cx + dy)$$

Main point:

- this matrix-multiplication framework does not handle translations; those must be handled as a separate operation, vector addition

Q: How to unify translations with rotations and scaling?

Answer: homogeneous coordinates



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_0 & a_1 \\ b_0 & b_1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_2 \\ b_2 \end{bmatrix} \quad \Leftrightarrow \quad \underbrace{\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ x \\ 1 \end{bmatrix}}_{\text{affine transformation in homogeneous coordinates}}$$

credit: google images

In general, 3 types of linear transformation:

- Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Scaling

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

credit: http://gra-phic.blogspot.com/2011/04/computer-graphics-and-geometric.html

Same idea in 3-d (but can't visualize)

Translation matrix in 3d:

$$T_{\mathbf{v}}\mathbf{p} = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + v_x \\ p_y + v_y \\ p_z + v_z \\ 1 \end{bmatrix} = \mathbf{p} + \mathbf{v}$$

credit: google images

In general:

$\mathbf{T}_{\boldsymbol{v}}$: Translation by an offset vector $\boldsymbol{v}$ $(x_v, y_v, z_v) = \begin{bmatrix} 1 & 0 & 0 & x_v \\ 0 & 1 & 0 & y_v \\ 0 & 0 & 1 & z_v \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$\mathbf{R}_\theta(x)$: Rotation by an angle θ about the x-axis $= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

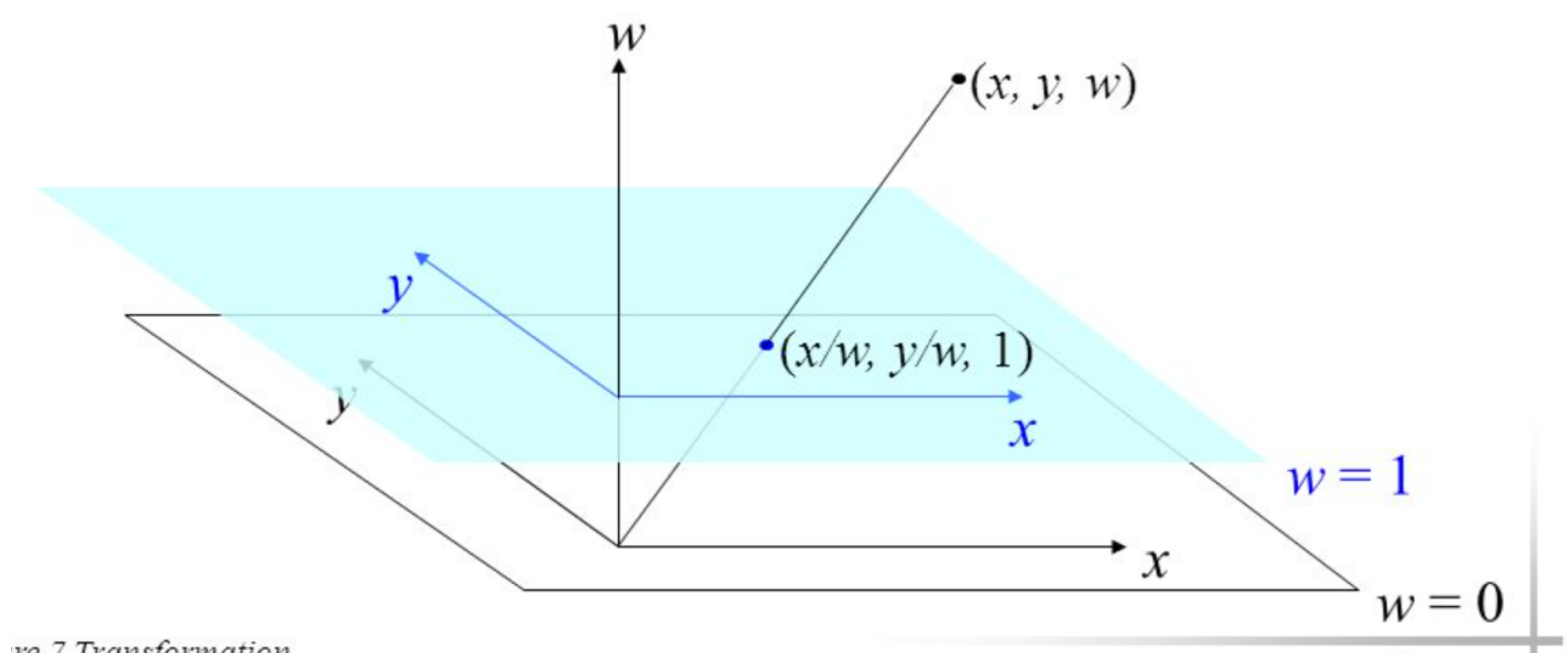
$\mathbf{R}_\theta(y)$: Rotation by an angle θ about the y-axis $= \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$\mathbf{R}_\theta(z)$: Rotation by an angle θ about the z-axis $= \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$\mathbf{S}_k$: Scaling by factors $k_x, k_y, k_z = \begin{bmatrix} x_k & 0 & 0 & 0 \\ 0 & y_k & 0 & 0 \\ 0 & 0 & z_k & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Last point: the w-coordinate

- For points in space, w = non-zero (usually 1)
  - e.g., (x,y,1)
- For directions, w = zero
  - e.g., (x,y,0)
  - for 2-d this corresponds to a vector *in* the viewing plane
  - you might run into this in the context of dealing with normals, which are directions rather than locations in space



credit: http://slideplayer.com/slide/3925951/