Geometric

- **Geometric Transformations**
    - Why do we use geometric transformations?
        - Bring multiple images into the same frame of reference (mosaicing)
        - When comparing images taken at different times
        - To remove distortion created by a lens (e.g. barrell distortion from wide angle lens) to give evenly spaced pixels
        - Simplifies any further processing e.g. OCR
    - **Problem Specification**
        - Given a distorted image **f(i , j)** and a corrected image **f(i' , j')**
        - **We model the transformations between their coordinates as:**
            - **$i = T_i(i' , j')$ and $j = T_j(i' , j')$**
            - **That is: given the coordinates (i' , j') in the corrected image, the functions $T_i()$ and $T_j()$ compute the corresponding (i , j) coordinates in the distorted image.**
            - It works in reverse/backwards
        - So to apply the transformation we need some information to define the transformation
            - This can be known in advance
            - Or can be determined through pairs of corresponding points between the two images
        - **Two main Scenarios for determining the correspondence:**
            - Obtaining the distorted image from a known pattern, so the corrected image is produced from this pattern (**image to known**)
            - Obtaining two images of the same object, where one image (**the distorted image**) is to be mapped into the frame of reference of the other image (**corrected image**) (**image to known**)
        - Once we have determined the transformation, we can apply it:
            - For every point in the output image
                - Determine where it came from using T
                - Interpolate a value for the output point
        - **Why does this transformation work in reverse?**
            - If we did the traditional input -> output , then in cases of image expansion, there would be gaps in the output
            - To avoid this, we consider every point in the output image, and compute a value for that point - no gaps

- **Types of Geometric Transformations**
    - **Affine Transformations**
        - Definition:
            - $$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$
        - **Unknown affine transformations require at least 3 observations**
            - Given 3 observations...
                - $$\begin{gathered} (i_1, j_1) \leftrightarrow (i'_1, j'_1) \\ (i_2, j_2) \leftrightarrow (i'_2, j'_2) \\ (i_3, j_3) \leftrightarrow (i'_3, j'_3) \end{gathered}$$
                - The greater number of observations the greater the accuracy
            - We can reorganise…
                - $$\begin{bmatrix} i_1 \\ j_1 \\ i_2 \\ j_2 \\ i_3 \\ j_3 \end{bmatrix} = \begin{bmatrix} i'_1 & j'_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & i'_1 & j'_1 & 1 \\ i'_2 & j'_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & i'_2 & j'_2 & 1 \\ i'_3 & j'_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & i'_3 & j'_3 & 1 \end{bmatrix} \begin{bmatrix} a_{00} \\ a_{01} \\ a_{02} \\ a_{10} \\ a_{11} \\ a_{12} \end{bmatrix}$$
            - **And take the inverse to compute the coefficients** (a values)
                - Multiply both sides by the inverse of the square matrix
                - **Given >3 observations, the matrix will not be square so we use the <u>psuedo inverse</u>**
        - **Known affine transformations**
            - **Translation**
                - $$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} 1 & 0 & m \\ 0 & 1 & n \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$
            - **Rotation**
                - $$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$
            - **Change of Scale**

-
$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

- **Skewing**
    -
    $$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} 1 & \tan\phi & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$
- **Panoramic Distortion**
    -
    $$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} a & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

- **Perspective Transformations**
    - All images are created through perspective projection (rays of light enter the camera through a single pinhole or lens)
    - **Perspective Transformations are needed when a planar surface lies in a plane which is not parallel to the image plane**
    -
    $$\begin{bmatrix} i.w \\ j.w \\ w \end{bmatrix} = \begin{bmatrix} p_{00} & p_{01} & p_{02} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & 1 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$
        - More complex than the affine transformation
    - **We require at least 4 mappings/observations**
    - From the above definition, we know…
        -
        $$i.w = p_{00}.i' + p_{01}.j' + p_{02}$$
        $$w = p_{20}.i' + p_{21}.j' + 1$$
        - Hence...
        $$i = p_{00}.i' + p_{01}.j' + p_{02} - p_{20}.i.i' - p_{21}.i.j'$$
        - And similarly...
        $$j = p_{10}.i' + p_{11}.j' + p_{12} - p_{20}.j.i' - p_{21}.j.j'$$
    - **Using our 4 mappings then we get…**

        - $$\begin{bmatrix} i_1 \\ j_1 \\ i_2 \\ j_2 \\ i_3 \\ j_3 \\ i_4 \\ j_4 \end{bmatrix} = \begin{bmatrix} i'_1 & j'_1 & 1 & 0 & 0 & 0 & -i_1 i'_1 & -i_1 j'_1 \\ 0 & 0 & 0 & i'_1 & j'_1 & 1 & -j_1 i'_1 & -j_1 j'_1 \\ i'_2 & j'_2 & 1 & 0 & 0 & 0 & -i_2 i'_2 & -i_2 j'_2 \\ 0 & 0 & 0 & i'_2 & j'_2 & 1 & -j_2 i'_2 & -j_2 j'_2 \\ i'_3 & j'_3 & 1 & 0 & 0 & 0 & -i_3 i'_3 & -i_3 j'_3 \\ 0 & 0 & 0 & i'_3 & j'_3 & 1 & -j_3 i'_3 & -j_3 j'_3 \\ i'_4 & j'_4 & 1 & 0 & 0 & 0 & -i_4 i'_4 & -i_4 j'_4 \\ 0 & 0 & 0 & i'_4 & j'_4 & 1 & -j_4 i'_4 & -j_4 j'_4 \end{bmatrix} \begin{bmatrix} p_{00} \\ p_{01} \\ p_{02} \\ p_{10} \\ p_{11} \\ p_{12} \\ p_{20} \\ p_{21} \end{bmatrix}$$
        - **Again, we multiply by the inverse of the square matrix**
            - If we have more than 4 observations, we use the psuedo inverse also
    - **More complex transformations**
        - Used in medical imaging with two images taken at different times/with different sensors
        - Approximation by a polynomial

          $$i = T_i(i',j') = a_{00} + a_{10}i' + a_{01}j' + a_{11}i'j' + a_{02}(j')^2 + a_{20}(i')^2 + a_{12}i'(j')^2 + a_{21}(i')^2 j' + a_{22}(i')^2 (j')^2 + \ldots$$

          $$j = T_j(i',j') = b_{00} + b_{10}i' + b_{01}j' + b_{11}i'j' + b_{02}(j')^2 + b_{20}(i')^2 + b_{12}i'(j')^2 + b_{21}(i')^2 j' + b_{22}(i')^2 (j')^2 + \ldots$$

            - No. of correspondences.
            - Distribution of points
            - Solve simultaneous linear equations
            - Least squared error solution

          Even more complex
            - Partition the image.
            - **The number of observations required is half of the number of terms in the polynomial**
            - If a geometric transformation is too complex for this, the image can be partitioned, with a transformation determined for each partition
- **Brightness interpolation**
    - Locations in the corrected image map back to ***real coordinates*** in the distorted image, so there is little chance that the mapping relates directly to one pixel
    - So the value at each pixel in the corrected image is interpolated
    - Three schemes…
    - **Nearest Neighbour Interpolation**
        - $$f'(i',j') = f(round(i), round(j))$$
        - Just round the real coordinate value so we take the value of the nearest pixel

- This gives us blocky effects - rarely used
- **Bilinear Interpolation**
  - $$f'(i',j') = (trunc(i)+1-i)(trunc(j)+1-j)f\big(trunc(i), trunc(j)\big) + \big(i-trunc(i)\big)(trunc(j)+1-j)f\big(trunc(i)+1, trunc(j)\big) + (trunc(i)+1-i)\big(j-trunc(j)\big)f(trunc(i), trunc(j)+1) + \big(i-trunc(i)\big)\big(j-trunc(j)\big)f(trunc(i)+1, trunc(j)+1)$$
  - **Assumes the brightness function is bilinear**
  - Combines the four nearest pixels brightness values using a weighting scheme
    - **Weighted by their distance to the true point (i , j)**
    - 2 weights to give the inverse distance
  - **Blurs the image**
- **Bicubic Interpolation**
  - Approximate the brightness value using a bi-cubic polynomial surface
  - Accounts for the values of **16 neighbouring pixels**, so **no blurring or stepping**
  - **Similar to Laplacian**
- **Camera Models - Removing Distortion**
  - **Radial Distortion**
    - Radially symmetric distortion where the level of distortion is related to the distance from the optical axis of the camera **(caused by the lens)**
      - **Barrel distortion -** Radial distortion in which the magnification decreases as distance increases
      - **Pincushion distortion** - Radial distortion in which the magnification increases as distance increases.
    - Taking the origin of the image is f(i,j)
    - $$i' = i(1 + k_1r^2 + k_2r^4 + k_3r^6)$$ $$j' = j(1 + k_1r^2 + k_2r^4 + k_3r^6)$$ $$r = \sqrt{i^2 + j^2}$$
  - **Tangential Distortion**
    - Uneven magnification from one side to the other
    - Lens not parallel to image plane
    - Again assuming the centre is f(i , j)
    - $$i' = i + \big(2p_1ij + p_2(r^2 + 2i^2)\big)$$ $$j' = j + \big(2p_2ij + p_1(r^2 + 2j^2)\big)$$
      - Where p_1 , p_2 are parameters describing the distortion
  - **Removing Distortion**
    - To determine the parameters creating the distortion in an imaging system, we must calibrate it

- Typically involves determining the camera model and distortion simultaneously
- Provide a known object at different positions and orientations
- **Computes a camera matrix and distortion parameters**