MATHEMATICS

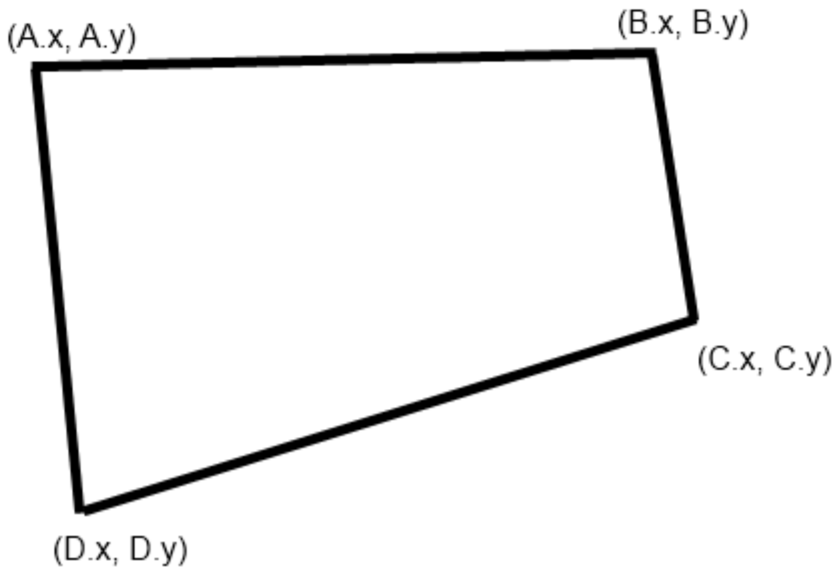
Mapping Irregular Quadrilateral to a Rectangle

Asked 11 years, 6 months ago Modified 4 months ago Viewed 40k times

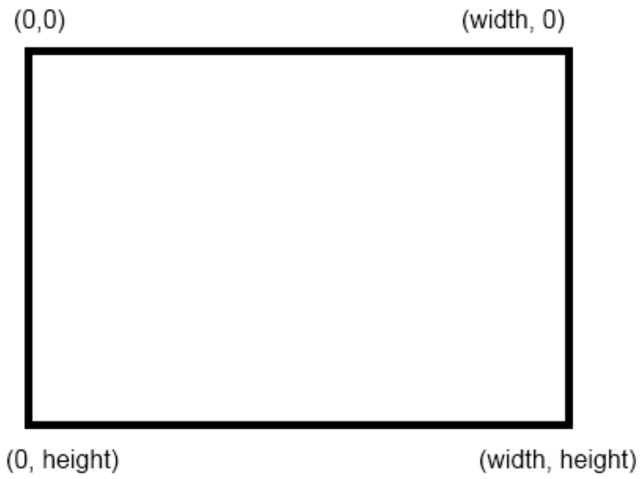
42

30

I have a camera looking at a computer monitor from varying angles. Since the camera is a grid of pixels, I can define the bounds of the monitor in the camera image as:



I hope that makes sense. What I want to do is come up with an algorithm to translate points within this shape to this:



I have points within the same domain as ABCD, as determined from the camera, but I need to draw these points in the domain of the monitor's resolution.

Does that makes sense? Any ideas?

geometry

Share Cite Follow

asked Dec 7, 2010 at 18:41
bufferz
521 1 5 4

You're asking for an *affine* transformation? – J. M. ain't a mathematician Dec 7, 2010 at 18:43

Yes, that's quite close to what I'm looking for. I'm looking to map a 4 sided shape with corners ABCD to A'B'C'D'. Is there a way to do this in general, or do i need to figure out the individual transform, rotate, shear, etc. differences between the two shapes? – bufferz Dec 7, 2010 at 18:56

I have a hard time understanding what you are asking. It seems you have an arbitrary quadrilateral and you want to map it to a rectangle? The most straight-forward way would be to average the opposite sides and take these to be your widths and heights. Is there something you want to be preserved by this transformation, e.g. area? – WWright Dec 7, 2010 at 19:17

Not all rectangles are achievable by pointing a camera at a rectangle. So ignore one of the corner-points and then do a linear transform (along with a translation to match up the origin) to match up the remaining triangle. The fourth point will magically end up where it should be. I would make this an answer, but I can't be bothered to actually work out which matrix I need. – Oscar Cunningham Dec 7, 2010 at 19:25

1 A 2D->2D affine transformation will not work, as they map parallelograms to parallelograms. You can do 3D->2D taking advantage of the depth to account for the non-parallelism (the two sides will be parallel in 3 space). If you know the depth, you can just follow the Wikipedia article and solve for the matrix elements of the transform. – Ross Millikan Dec 7, 2010 at 19:25

11 Answers

Sorted by: Highest score (default)

21

In general there is no affine transformation that maps an arbitrary quadrangle onto a rectangle. But there is (exactly one) projective transformation T that maps a given quadrangle (A, B, C, D) in the projective plane onto a given quadrangle $(A', B', C' D')$ in the same or another projective plane. This T is *collinear*, i.e., it maps lines to lines. To do the calculations you have to introduce homogeneous coordinates (x, y, z) such that $D = (0, 0, 1)$, $C = (1, 0, 1)$, $A = (0, 1, 1)$, $B = (1, 1, 1)$ and similarly for A', B', C', D'. With respect to these coordinates the map T is linear and its matrix is the identity matrix.

Share Cite Follow

answered Dec 7, 2010 at 19:33
Christian Blatter
217k 13 167 426

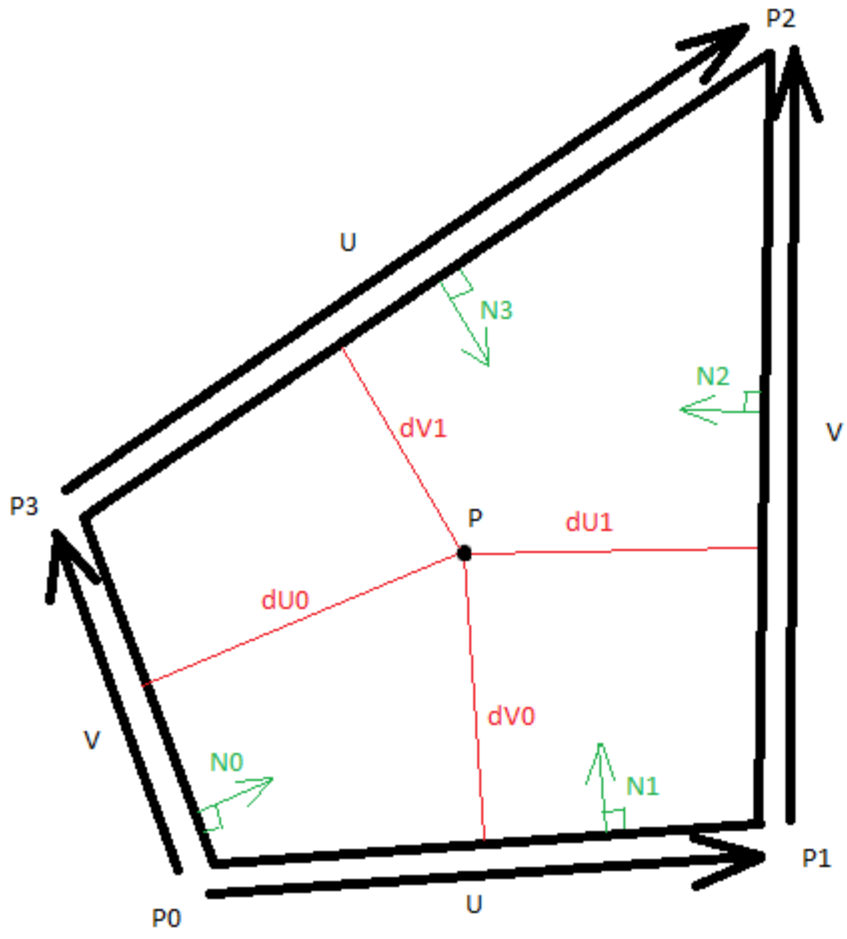
1 I'm a bit rusty on my matrix math. I'm not looking for a handout algorithm but I'm wondering if there's a wikipedia or similar article regarding your suggestion. I'm trying to end up with a function relating a ABCD and A'B'C'D' directly, so I can implement that into my software project. – bufferz Dec 8, 2010 at 2:43

4 It cannot be done without some matrix manipulation. The following page seems to contain very explicit formulas pertaining to your problem: geometrictools.com/Documentation/PerspectiveMapping.pdf – Christian Blatter Dec 8, 2010 at 20:04

1 The link has changed to geometrictools.com/Documentation/PerspectiveMappings.pdf – VoteCoffee Oct 22, 2014 at 16:45

16

The best solution I've found so far on a forum lost in the sea of forums is to decompose your problem like this :



Here, U and V represent coordinates within the quadrilateral (scaled between 0 and 1).

From $P0$, $P1$, $P2$ & $P3$ we can easily compute the normalized normal vectors $N0$, $N1$, $N2$ & $N3$. Then, it's easy to see that :

$$u = \frac{dU0}{dU0 + dU1} = \frac{(P - P0) \cdot N0}{(P - P0).\,N0 + (P - P2) \cdot N2}$$
$$v = \frac{dV0}{dV0 + dV1} = \frac{(P - P0) \cdot N1}{(P - P0).\,N1 + (P - P3) \cdot N3}.$$

This parametrization works like a charm and is really easy to compute within a shader for example. What's tricky is the reverse: finding $P(x, y)$ from (u, v) so here is the result:

$$x = \frac{vKH \cdot uFC - vLI \cdot uEB}{vJG \cdot uEB - vKH \cdot uDA},$$
$$y = \frac{vLI \cdot uDA - uFC \cdot vJG}{vJG \cdot uEB - vKH \cdot uDA},$$

where:

$$uDA = u \cdot (D - A), \quad uEB = u \cdot (E - B), \quad uFC = u \cdot (F - C),$$
$$vJG = v \cdot (J - G), \quad vKH = v \cdot (K - H), \quad vJG = v \cdot (J - G),$$

and finally:

$$\begin{aligned} A &= N0_x, & B &= N0_y, & C &= -P0 \cdot N0, \\ D &= N0_x + N2_x, & E &= N0_y + N2_y, & F &= -P0 \cdot N0 - P2 \cdot N2, \\ G &= N1_x, & H &= N1_y, & I &= -P0 \cdot N1, \\ J &= N1_x + N3_x, & K &= N1_y + N3_y, & L &= -P0 \cdot N1 - P2 \cdot N3. \end{aligned}$$

I've been using this successfully for shadow mapping of a deformed camera frustum mapped into a regular square texture and I can assure you it's working great! :D

Share Cite Follow

edited Apr 8, 2014 at 17:44
TDN169
115 5

answered Feb 1, 2012 at 13:18
Patapom
161 1 2

1 What are u, v? – nbubis Mar 21, 2014 at 0:21

How do you know P? It seems placed in an arbitrary position – Pedro Batista Jan 5, 2017 at 17:20

Thank you! How will this perform if P is outside of the quadrilateral? – seth10 Jan 18, 2017 at 21:24

Excellent - this should be the accepted answer - "..works like a charm.." indeed! – slashmais Mar 19, 2017 at 11:21

This is great, although it becomes discontinuous when P is outside of the quadrilateral. – Tim MB Nov 8, 2017 at 21:05

2

Try this solution, it worked for me.

Share Cite Follow

answered Jul 1, 2011 at 22:56
jbm
121 2

As far as I can tell the solution you point to fails if you have a parallelogram as an input. For example, try giving it a . It's quite possible I'm not understanding something about the solution but I'd love to know what it is if you can get it to work for parallelograms. – Mattia Nov 29, 2011 at 21:27

1 @Mattia The linked solution describes a method (bilinear interpolation) to map the unit square onto an arbitrary quadrilateral. There are no restrictions on the quadrilateral. This is probably exactly what the OP is looking for. If the inverse of this mapping is required, this is also discussed in the link. – yasmar Feb 1, 2012 at 14:09

2

Here is a solution implemented in VBA, a General Algebraic solution, more general than the augmented 2D affine transformation formulation on Wikipedia.

```
Function Quad_to_Logical_Cell(Qx() As Double, Qy() As Double, x As Double, y As Double) As
Variant

  'WJW 7-13-15
  'This function performs a coordinate transform from X,Y space to the normalized L,M.
  '
  'If a point {is within {0,1} on both axes, it is within the transformed unit square.
  'Qx,Qy vectors contain the 4 coordinates of the corners - x and y values, respectively,
ordered as indicated below:
  '
  'The unit cell L(l,m) corresponding to Q(x,y) is oriented as:
  'L0(x=0,y=0),L1(0,1), L2(1,1), L3(1,0).  The order matters.
  'The following represent an algebraic solution to the system:
  'l=a1 + b1x + c1y + d1xy
  'm=a2 + b2x + c2y + d2xy


    Dim L_Out() As Double
    ReDim L_Out(2)

    ax = (x - Qx(0)) + (Qx(1) - Qx(0)) * (y - Qy(0)) / (Qy(0) - Qy(1))
    a3x = (Qx(3) - Qx(0)) + (Qx(1) - Qx(0)) * (Qy(3) - Qy(0)) / (Qy(0) - Qy(1))
    a2x = (Qx(2) - Qx(0)) + (Qx(1) - Qx(0)) * (Qy(2) - Qy(0)) / (Qy(0) - Qy(1))
    ay = (y - Qy(0)) + (Qy(3) - Qy(0)) * (x - Qx(0)) / (Qx(0) - Qx(3))
    a1y = (Qy(1) - Qy(0)) + (Qy(3) - Qy(0)) * (Qx(1) - Qx(0)) / (Qx(0) - Qx(3))
    a2y = (Qy(2) - Qy(0)) + (Qy(3) - Qy(0)) * (Qx(2) - Qx(0)) / (Qx(0) - Qx(3))
    bx = x * y - Qx(0) * Qy(0) + (Qx(1) * Qy(1) - Qx(0) * Qy(0)) * (y - Qy(0)) / (Qy(0) -
Qy(1))
    b3x = Qx(3) * Qy(3) - Qx(0) * Qy(0) + (Qx(1) * Qy(1) - Qx(0) * Qy(0)) * (Qy(3) - Qy(0))
/ (Qy(0) - Qy(1))
    b2x = Qx(2) * Qy(2) - Qx(0) * Qy(0) + (Qx(1) * Qy(1) - Qx(0) * Qy(0)) * (Qy(2) - Qy(0))
/ (Qy(0) - Qy(1))
    by = x * y - Qx(0) * Qy(0) + (Qx(3) * Qy(3) - Qx(0) * Qy(0)) * (x - Qx(0)) / (Qx(0) -
Qx(3))
    b1y = Qx(1) * Qy(1) - Qx(0) * Qy(0) + (Qx(3) * Qy(3) - Qx(0) * Qy(0)) * (Qx(1) - Qx(0))
/ (Qx(0) - Qx(3))
    b2y = Qx(2) * Qy(2) - Qx(0) * Qy(0) + (Qx(3) * Qy(3) - Qx(0) * Qy(0)) * (Qx(2) - Qx(0))
```

Share Cite Follow

answered Jul 14, 2015 at 23:09

Wes

21 1

2

Take a look on Gernot Hoffmann's tutorial on image rectification. There are also the special cases (rectangle to quadrilateral) explained.

Another page that helped me discusses 2D perspective transformation (i.e. planar homography).

For a deep understanding of the topic and more numerically stable algorithms, I can only recommend *Hartley & Zisserman: Multi-View Geometry in Computer Vision*.

Share Cite Follow

edited Aug 24, 2017 at 6:45
hippietrail
427 2 12

answered Apr 10, 2012 at 0:24
Libor
1,267 1 13 25

1

I've been wrestling with a very similar problem in order to determine gradients in an irregular quad grid and needing to map points within arbitrary quadrilaterals to a unit square. In addition, I require the inverse mapping of the x and y axis at the mapped normalized coordinate location back into the quad so I can determine the orientation of the quad grid at that point. i.e. if `[x',y']` are the transformed coordinates, I need to be able to do an inverse transform on `[0,y'],[1,y']` and `[x',0],[x',1]`. Here is what I've come up with:

You can divide the quad into two tris, and use affine maps on these individually. This is not difficult. This will create a noticable effect at the division between the two tris, however.

If you want a smooth mapping from a quad to a square (or rectangle), you need to use a non-affine transform such as a projective transform. There are other transforms other than projective that will also work, and also be colinear (preserve straight lines).

If `[x1,y1],[x2,y2],[x3,y3],[x4,y4]` are the four points in the quad, then the 4x4 matrix **B** in the the following will yield a mapping into the square (on the RHS) that seems to work and may be easier to compute than the proper 3x3 projective matrix.

```
%     [x1 y1 x1*y1 1]              [0 0 0 1]
%     [x2 y2 x2*y2 1]   X  B  =    [1 0 0 1]
%     [x3 y3 x3*y3 1]              [0 1 0 1]
%     [x4 y4 x4*y4 1]              [1 1 1 1]
```

The question I have is that if one does this, and then wants to use the inverse of **B** to do the inverse transform, how do you calculate the third elements of the location vectors for the orthogonal coordinates. (They are no longer x*y.)

NOTE: If you want to map into any other (arbitrary) quadrilateral (such as a rectangle), then just replace the RHS of what I have above with the new coordinates.

```
%     [x1 y1 x1*y1 1]              [x1' y1' x1'*y1' 1]
%     [x2 y2 x2*y2 1]   X  B  =    [x2' y2' x2'*y2' 1]
%     [x3 y3 x3*y3 1]              [x3' y3' x3'*y3' 1]
%     [x4 y4 x4*y4 1]              [x4' y4' x4'*y4' 1]
```

Share Cite Follow   edited Dec 10, 2010 at 4:15   answered Dec 10, 2010 at 4:05
Grant

1

You could approach this by using an isoparametric mapping. Say the quadrilateral object is said to be in a $x_1 - y_1$ coordinate frame, while the rectangle is in a new $x_2 - y_2$ frame. What you can do is find $x_1 = x_1(x_2, y_2)$ and $y_1 = y_1(x_2, y_2)$ using an interpolation-based mapping.

Say we define each vertex as a 2D vector $\vec{P}_i$, we can end up with the following mapping to find an a given $\vec{P}$ as a function of x_2 and y_2:

$$\vec{P}(x_2, y_2) = \sum_{i=1}^{4} \vec{P}_i h_i(x_2, y_2)$$

Now, assume point A, $\vec{P}_1$, corresponds with $(0,0)$ location, point B, $\vec{P}_2$, corresponds with $(width, 0) = (w, 0)$, etc. With that, we can arrive at the following expressions for h_i:

$$h_1(x_2, y_2) = \frac{(x_2 - w)(y_2 - h)}{wh}$$

$$h_2(x_2, y_2) = \frac{x_2(h - y_2)}{wh}$$

$$h_3(x_2, y_2) = \frac{x_2 y_2}{wh}$$

$$h_4(x_2, y_2) = \frac{(w - x_2)y_2}{wh}$$

Using all this information, you could loop through the rectangle to find the $\vec{P}$ coordinate in the original image that each (x_2, y_2) pixel is associated with, then get the pixel information and drop it into the (x_2, y_2) pixel. As a note, the h_i expressions were found via Lagrangian interpolation procedures.

Share Cite Follow

answered Oct 17, 2014 at 0:29
spektr
549 3 11

0

You may find this [example Perl code]() useful, using the Imager library.

Share Cite Follow

answered Feb 6, 2014 at 16:00
Richard Fairhurst
101

0

Building off of @Patapom's answer, the goal is to find the **p** in image space corresponding to an arbitrary u,v. Starting from the transform:

$$u = \frac{(\mathbf{p}-\mathbf{p_0})\cdot\mathbf{n}_0}{(\mathbf{p}-\mathbf{p_0}).\mathbf{n}_0+(\mathbf{p}-\mathbf{p_2})\cdot\mathbf{n}_2}$$
$$v = \frac{(\mathbf{p}-\mathbf{p_0})\cdot\mathbf{n}_1}{(\mathbf{p}-\mathbf{p_0}).\mathbf{n}_1+(\mathbf{p}-\mathbf{p_3})\cdot\mathbf{n}_3}.$$

We can isolate **p**, and rewrite the equality as $A\mathbf{p} = \mathbf{b}$, where:

$$A \equiv \begin{bmatrix} u\mathbf{n}_2^\top - (1-u)\mathbf{n}_0^\top \\ v\mathbf{n}_3^\top - (1-v)\mathbf{n}_1^\top \end{bmatrix}$$

$$b \equiv \begin{bmatrix} u\mathbf{p}_2^\top\mathbf{n}_2 - (1-u)\mathbf{p}_0^\top\mathbf{n}_0 \\ v\mathbf{p}_3^\top\mathbf{n}_3 - (1-v)\mathbf{p}_0^\top\mathbf{n}_1 \end{bmatrix}$$

Since A is a 2x2 matrix it can analytically inverted to solve for **p**. Here's an example python routine:

```
def map_uv_to_xy(u, v, P, N):

    nu = 1 - u
    nv = 1 - v
    A_11 = u*N[2][0]-nu*N[0][0]
    A_12 = u*N[2][1]-nu*N[0][1]
    A_21 = v*N[3][0]-nv*N[1][0]
    A_22 = v*N[3][1]-nv*N[1][1]

    b_0 = u*(P[2][0]*N[2][0] + P[2][1]*N[2][1])-nu*(P[0][0]*N[0][0] + P[0][1]*N[0][1])
    b_1 = v*(P[3][0]*N[3][0] + P[3][1]*N[3][1])-nv*(P[0][0]*N[1][0] + P[0][1]*N[1][1])
    x = b_0* A_22 + b_1*-A_12
    y = b_0*-A_21 + b_1* A_11
    det_A = A_11*A_22 - A_12*A_21
    return x/det_A, y/det_A
```

Share Cite Follow

answered Apr 12, 2019 at 3:00
Michael Mason
1

0

I found this explanation easy to understand if you are familiar with basic linear algebra: http://faculty.salina.k-state.edu/tim/mVision/ImageFormation/homography.html

The basic idea is that you have 2x4 matrices $\mathbf{P}$ (coordinates of the four corners in the image you have) and $\mathbf{Q}$ (coordinates of the four corners in the projection you want, and you are trying to estimate $\mathbf{H}$ where $\mathbf{Q} = \mathbf{HP}$, using the pseudoinverse:

$$\mathbf{H} = \mathbf{QP}^T(\mathbf{PP}^T)^{-1}$$

or use a least-square estimator (like `numpy.linalg.lstsq` in Python) to solve $\mathbf{Q} = \mathbf{HP}$ for $\mathbf{H}$.

Share Cite Follow

answered Jan 9 at 20:50
Jason S
2,917 1 19 24

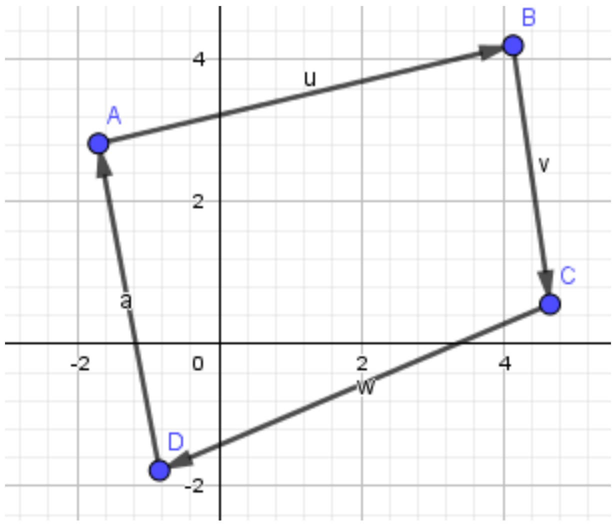
0

HINT

A, B, C, D are not in the same plane.

A very approximate rectangular projection ratio ... by area projections with extended boundary length), may be obtained considering boundary vector lenghts.

$$\frac{\frac{1}{2}(|u \times v| + |w \times a|)}{(|u| + |v| + |w| + |a|)^2}$$



The rectangle can be now re-sized.

Share Cite Follow

answered Jan 9 at 23:20
Narasimham
36.4k 7 34 88