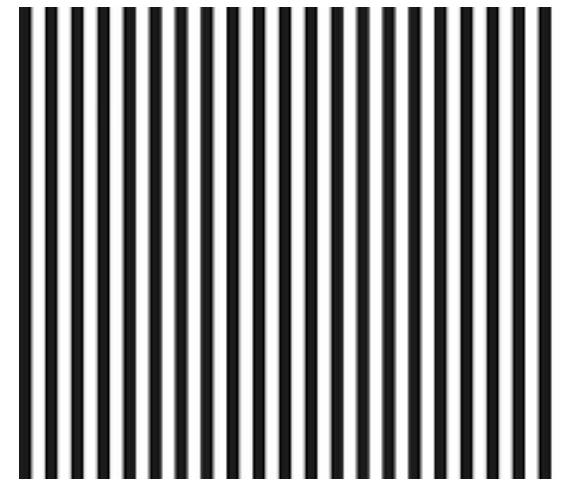
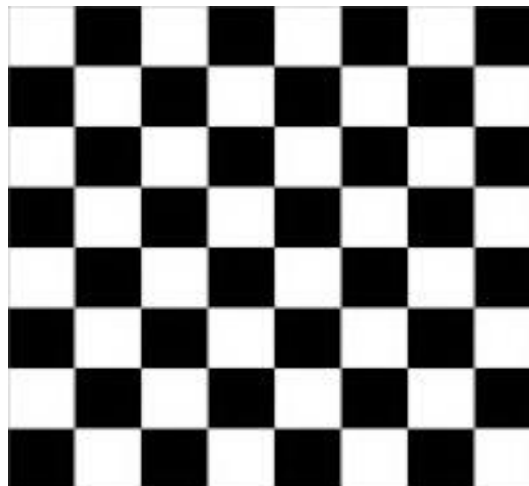
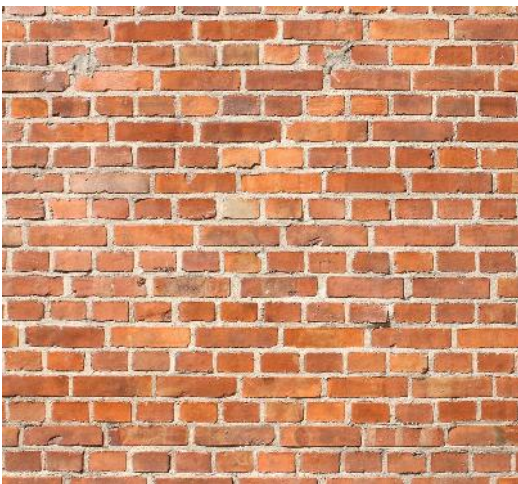
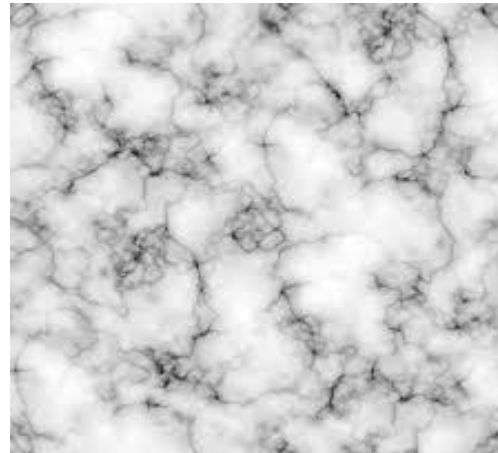


lecture 16

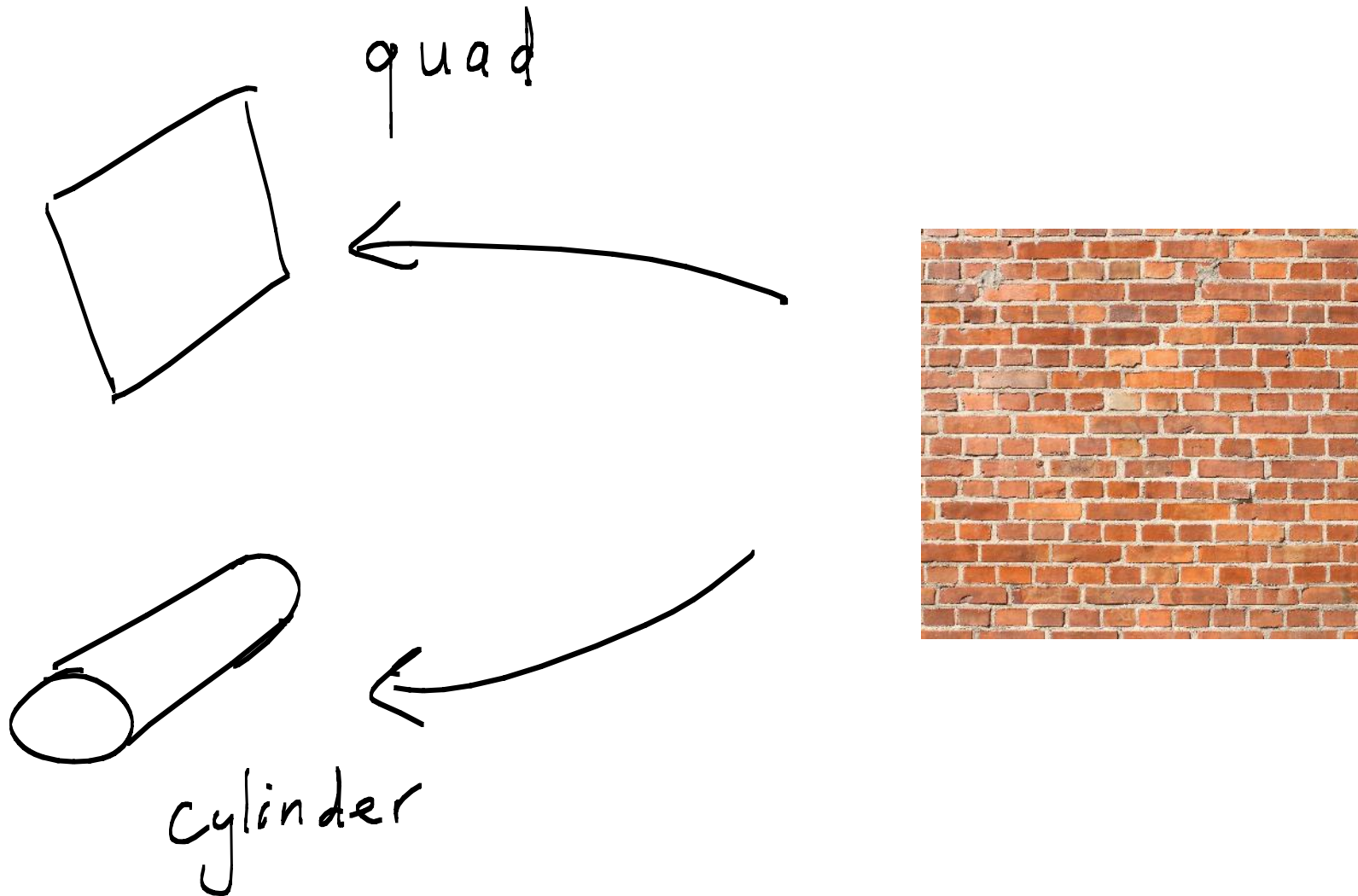
Texture mapping

Aliasing (and anti-aliasing)

Texture (images)

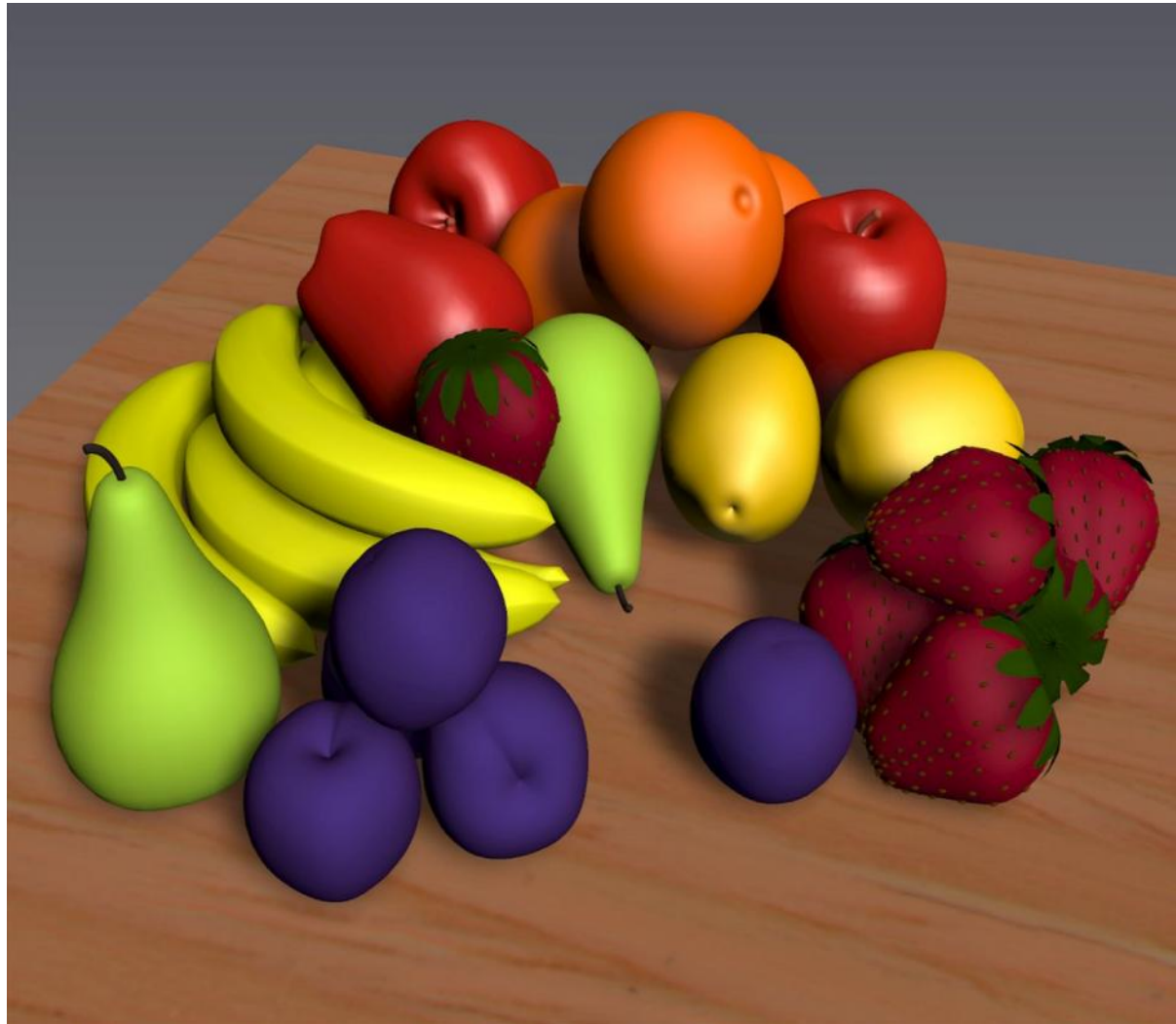


Texture Mapping



Q: Why do we need texture mapping ?

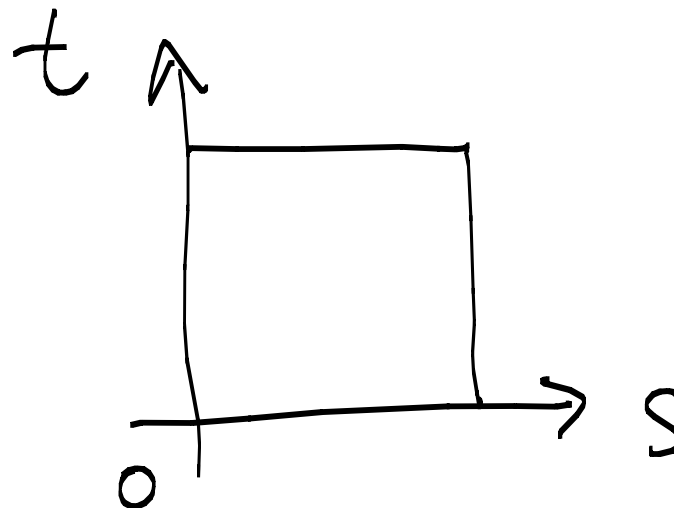
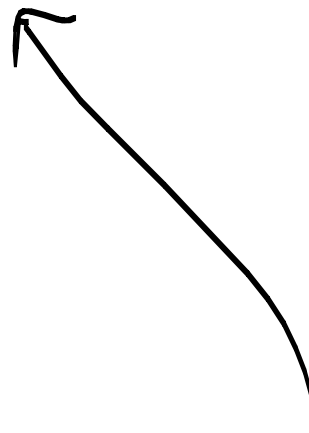
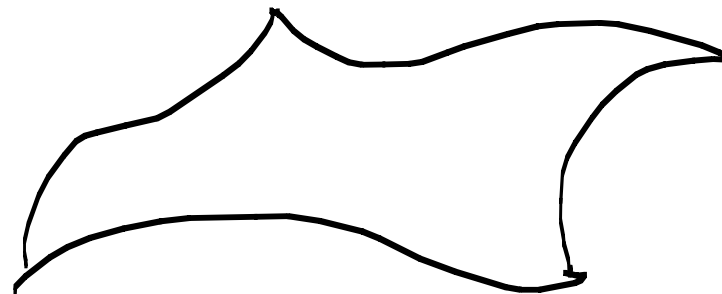
A: Because objects look fake and boring without it.



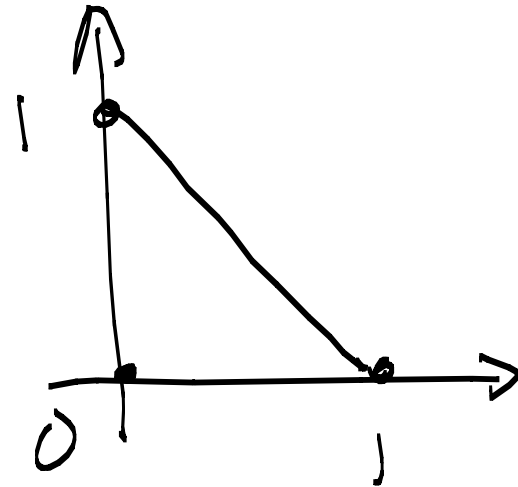
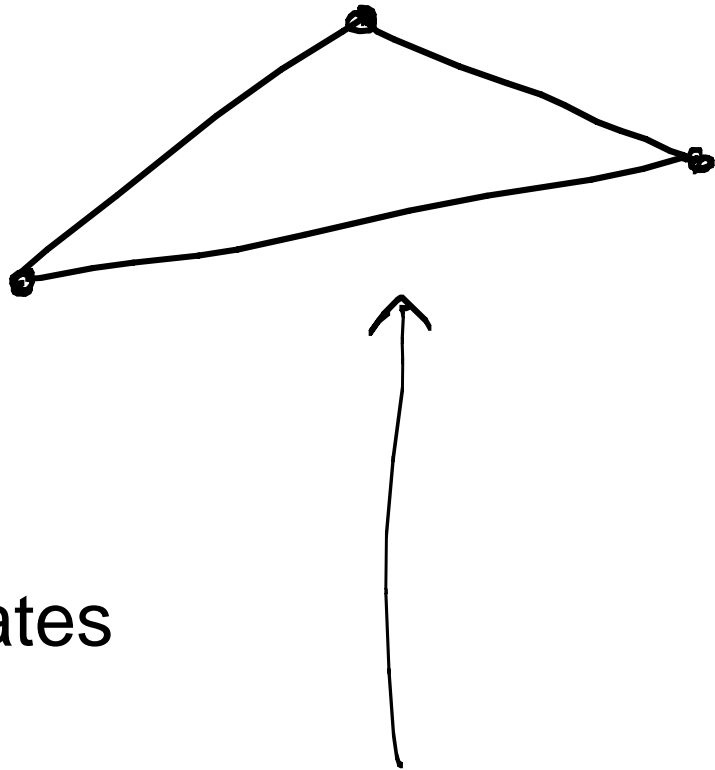
Adding texture improves realism.

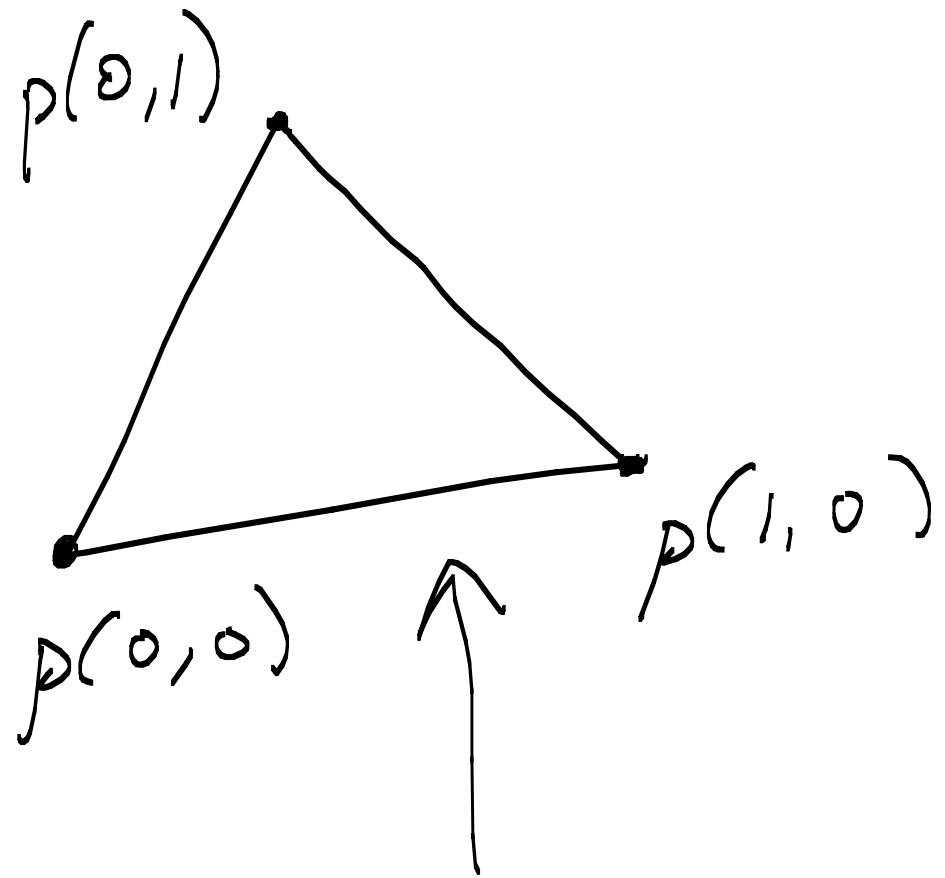


recall mapping
for bicubic patch

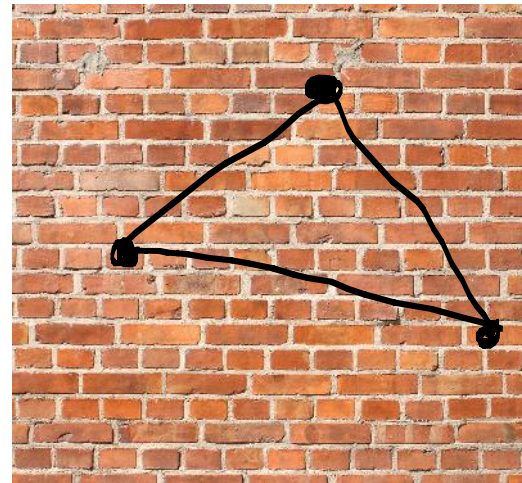


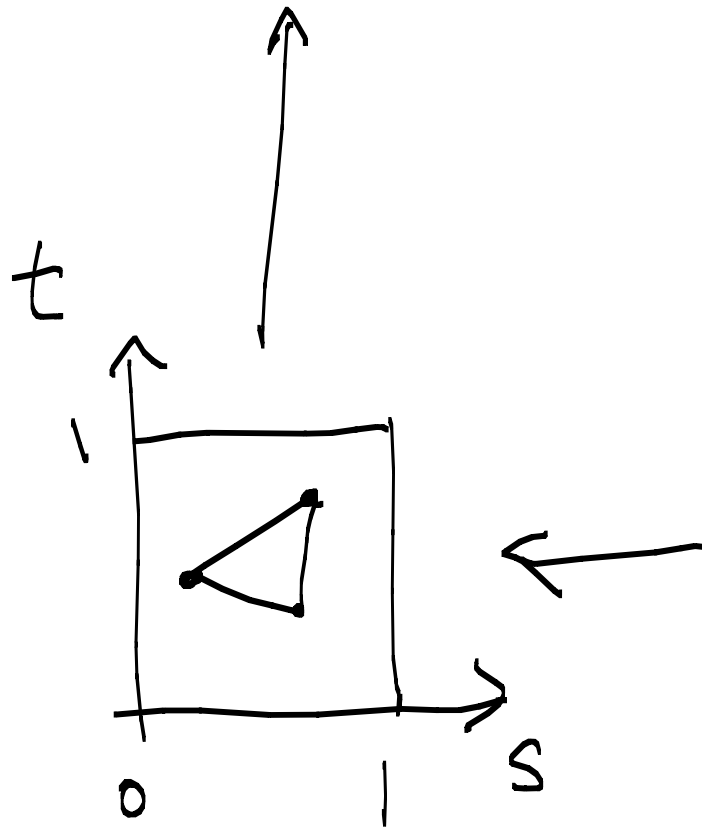
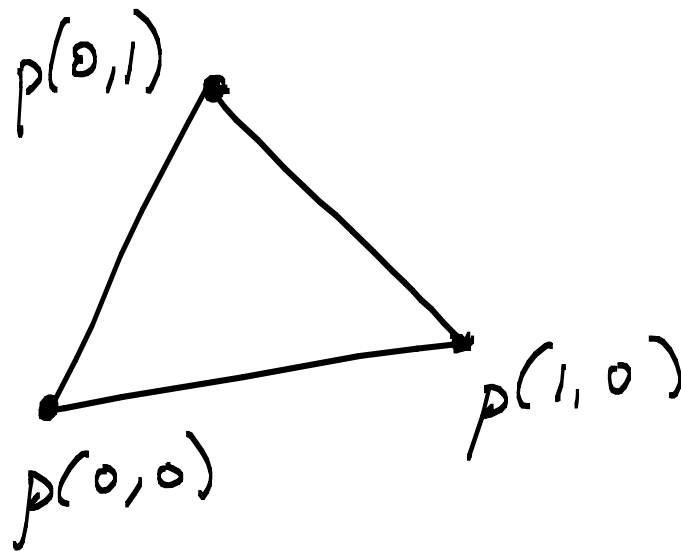
recall mapping
for barycentric coordinates
of a triangle





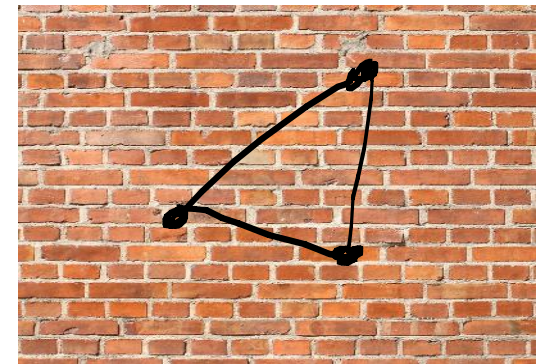
We would like to map the texture image coordinates to surface coordinates.





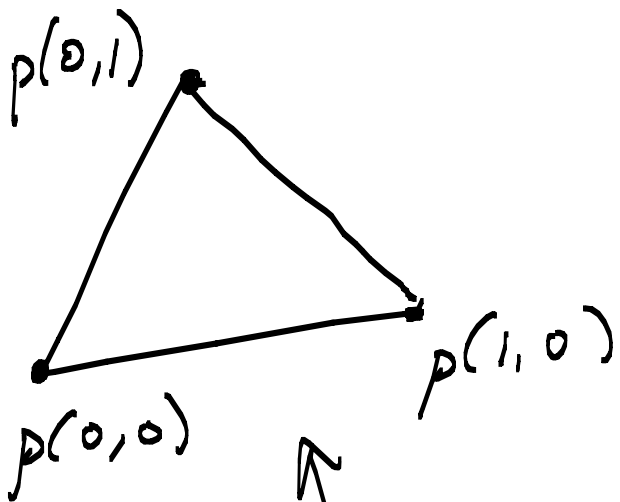
normalized texture
coordinates

$N_x \times N_y$



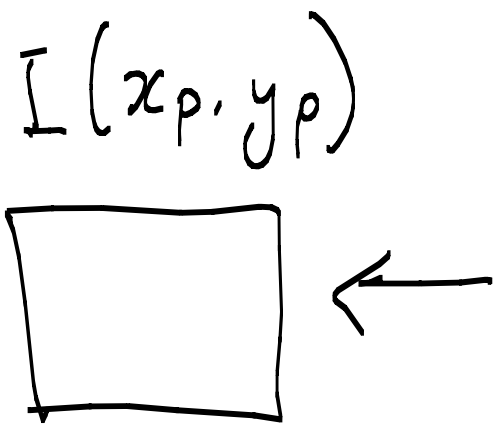
texture image
(not necessarily square)

We do so, we use
an intermediate
map to normalized
coordinates.

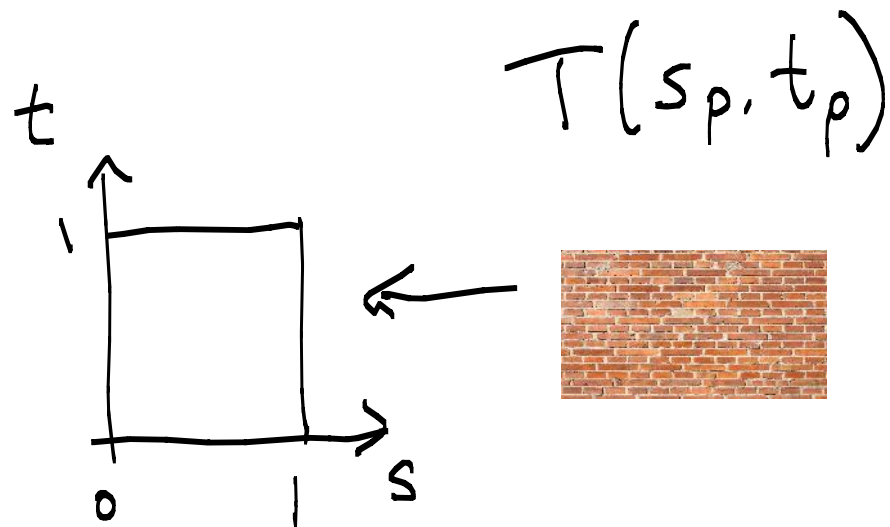
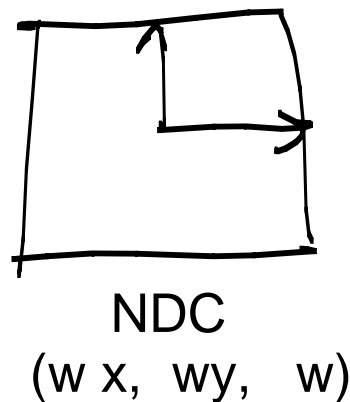


world to camera to
clip coordinates
(x and y only)

normalized texture coordinates
to world coordinates

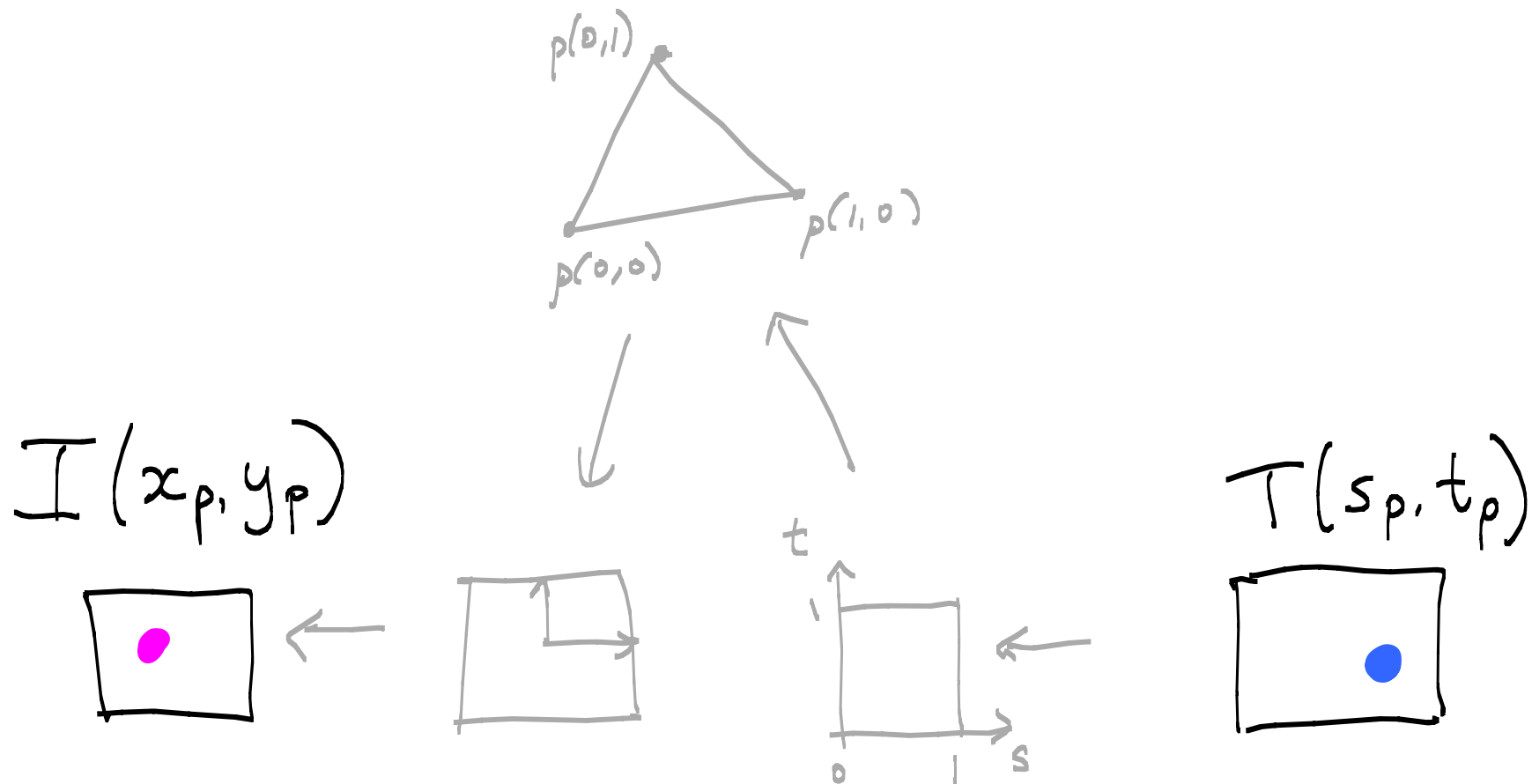


display
coordinates

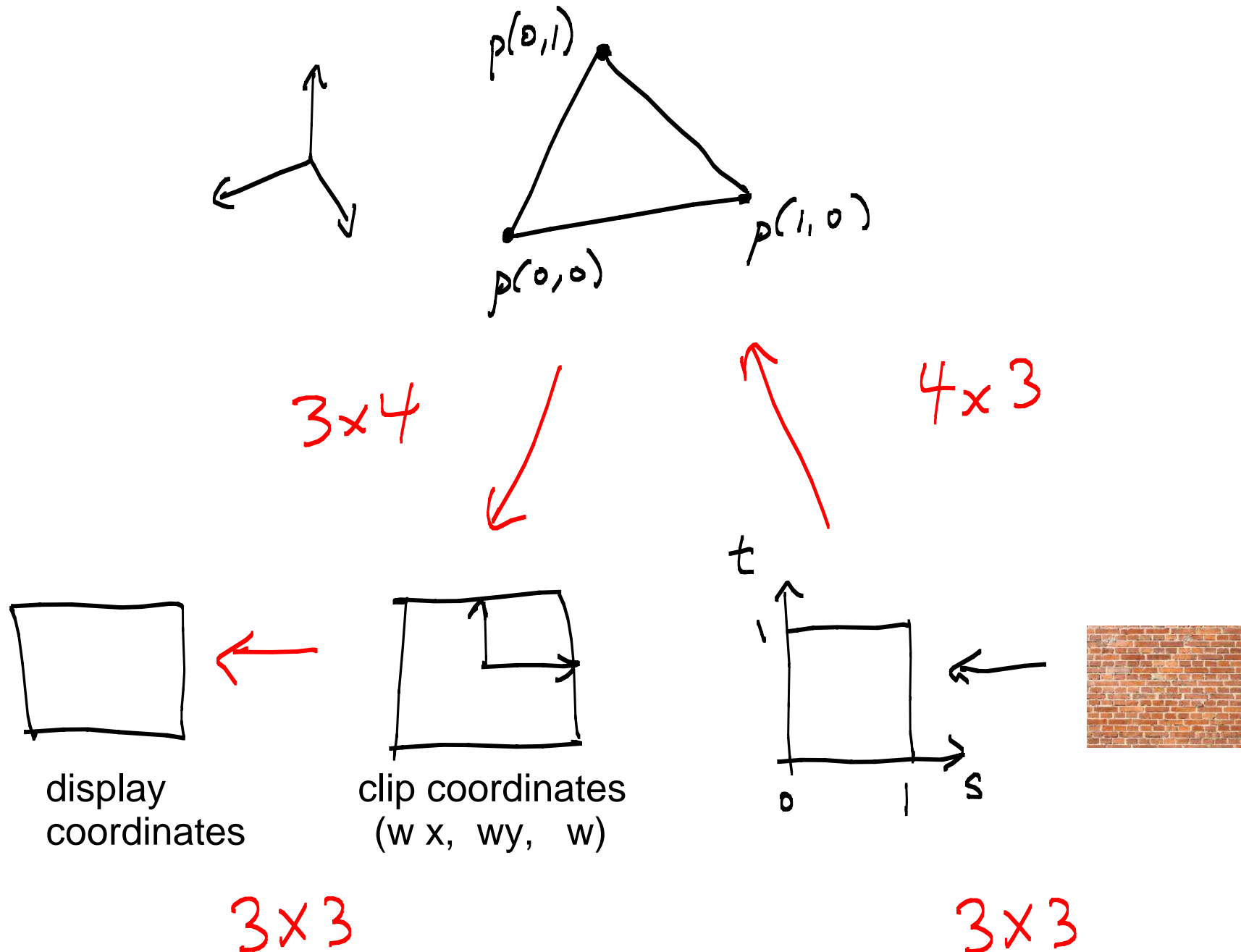


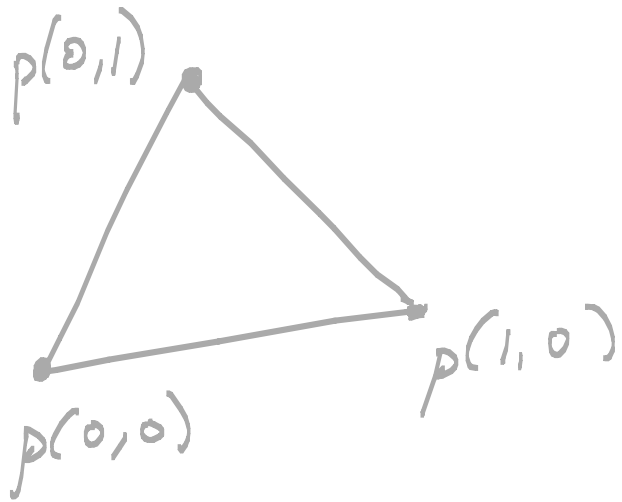
What is texture mapping?

```
for each pixel in the image projection of the polygon{  
  compute corresponding texel position  
  copy texture RGB to image pixel RGB  
}
```



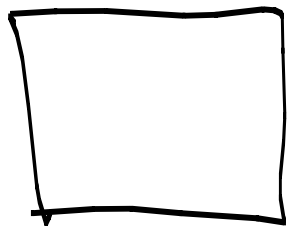
Let's think about the matrices that are used for this mapping.
Here we simplify: assume camera coords = world coords.



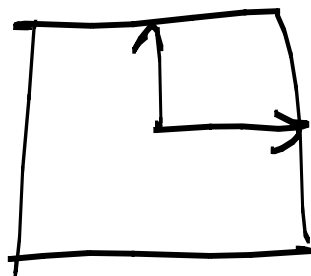


3×4

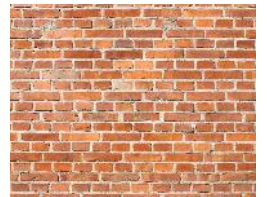
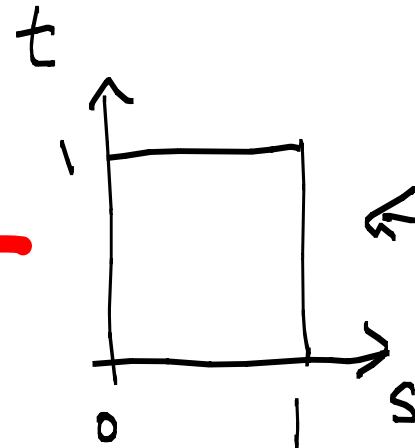
4×3



display
coordinates



clip coordinates
(w x, wy, w)



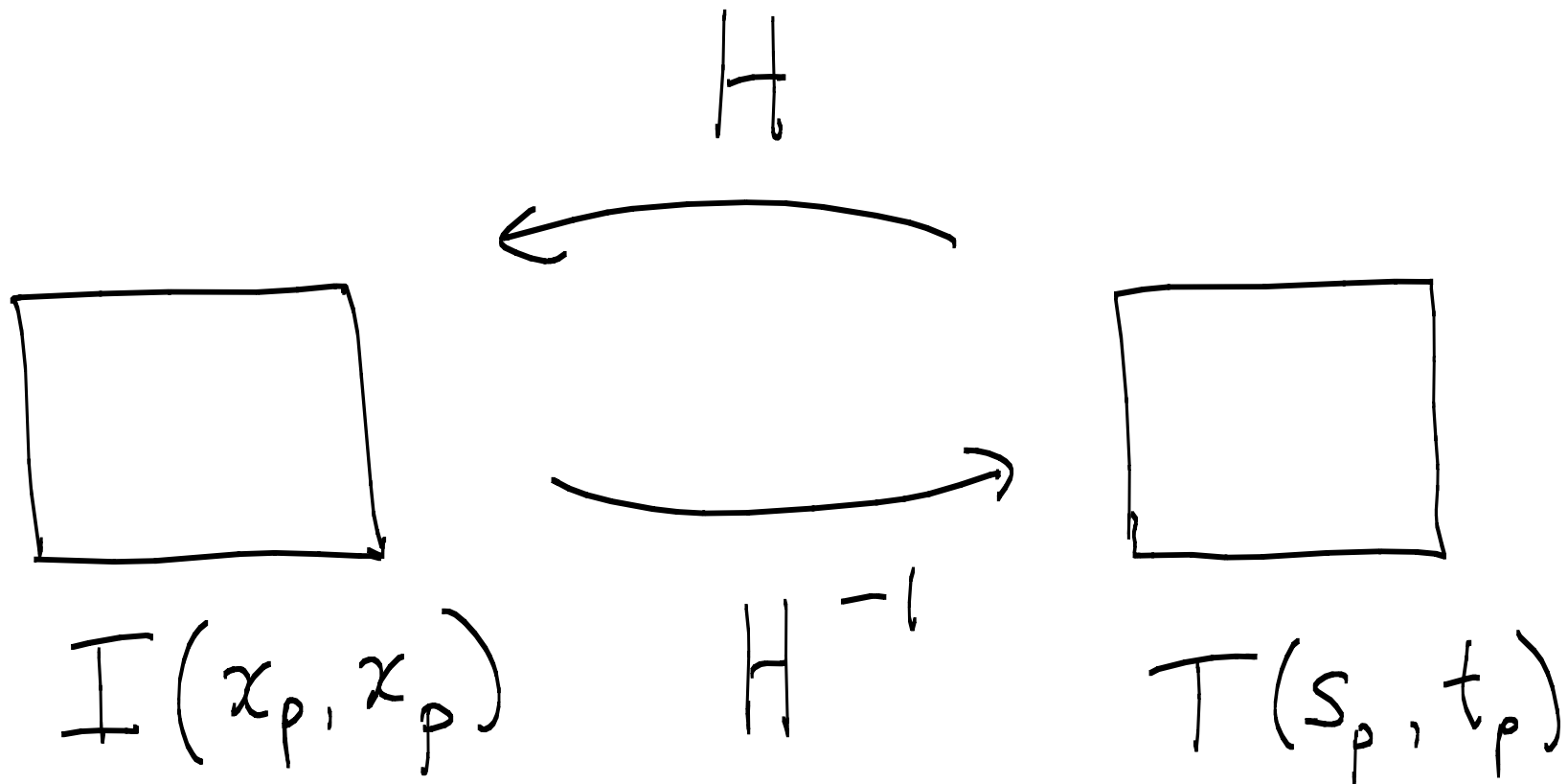
3×3

3×3

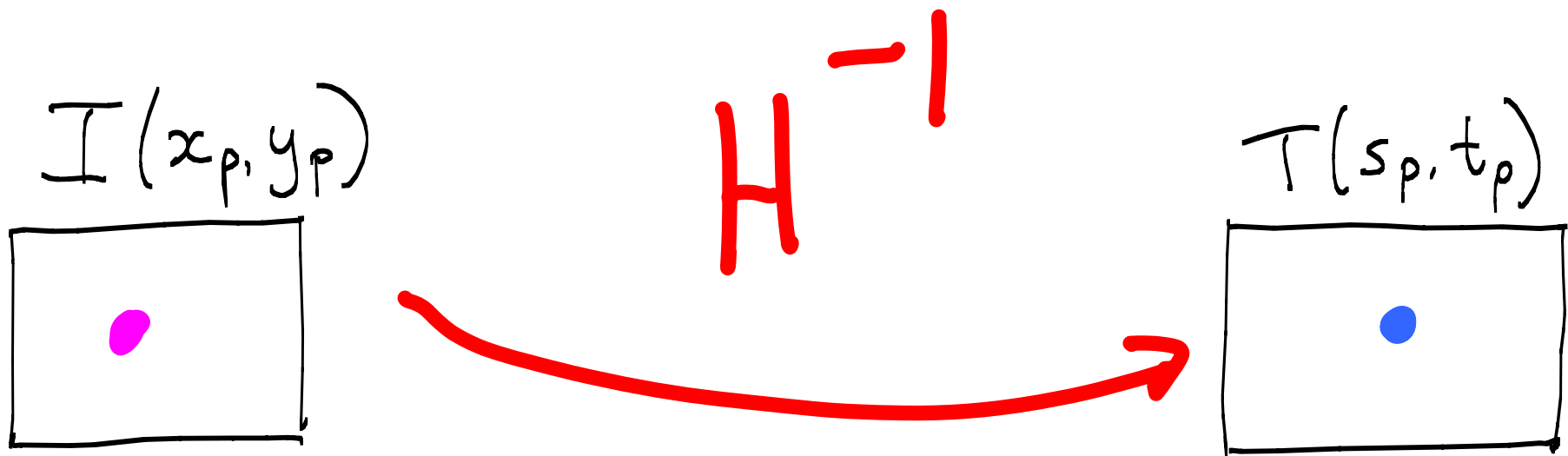
3×3

Homography

A homography is an invertible 3×3 matrix that maps between 2D spaces that are represented in homogeneous coordinates.



for each pixel in the image projection of the polygon{
 Use (inverse) homography to compute corresponding texel position
 Use texture RGB to determine image pixel RGB
}

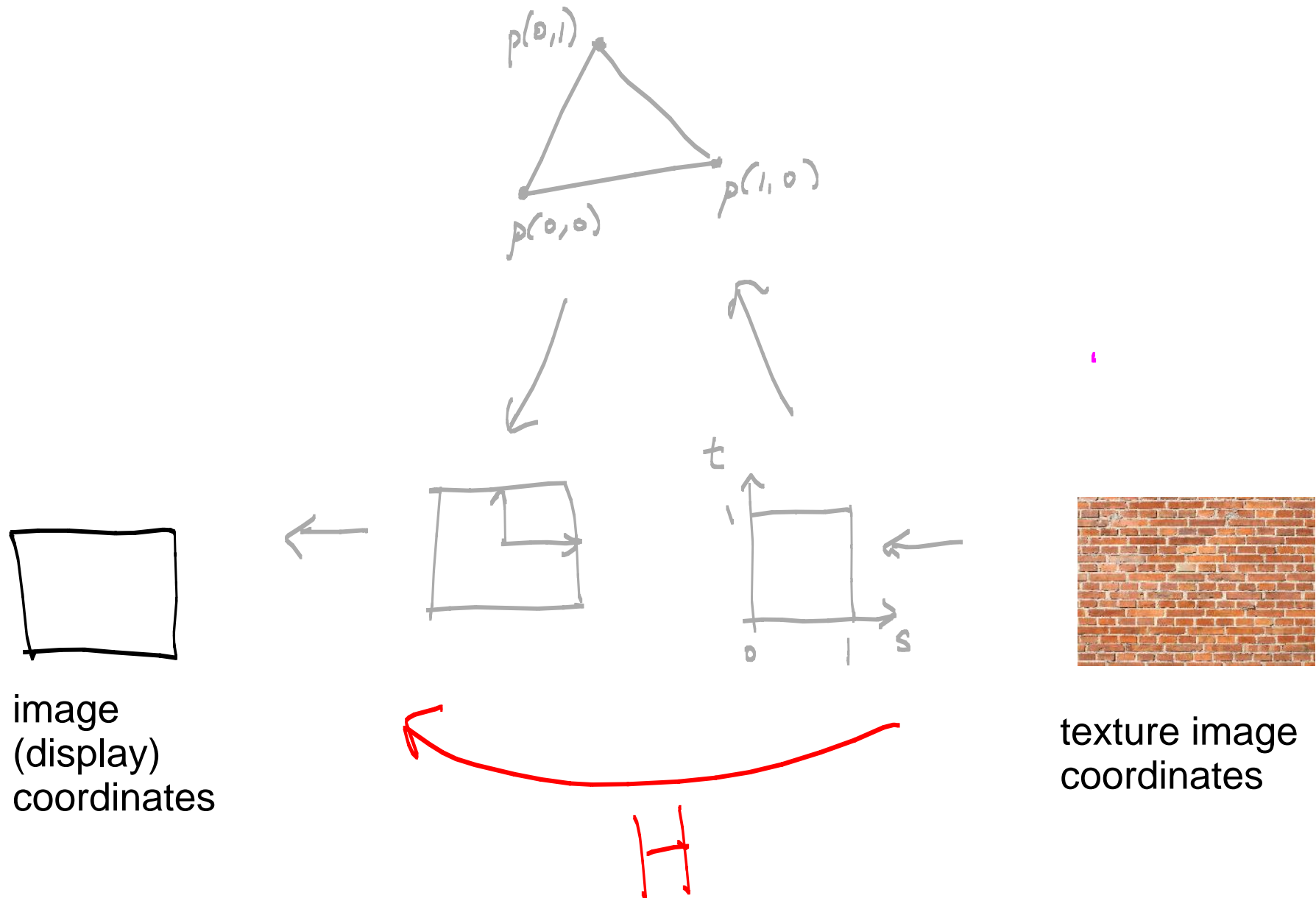


Details

How to construct the homography ?

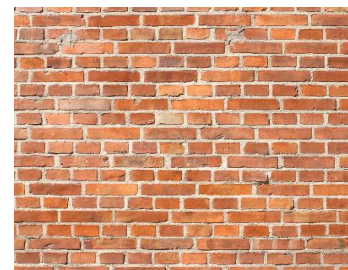
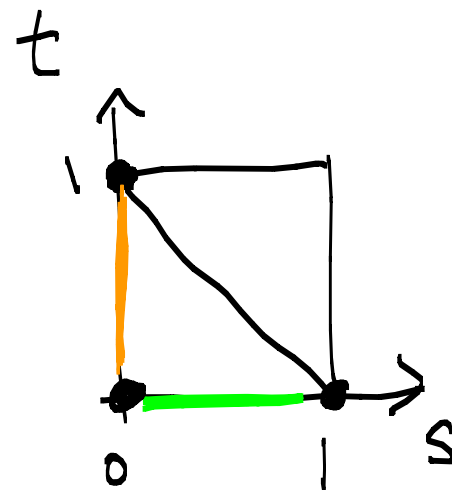
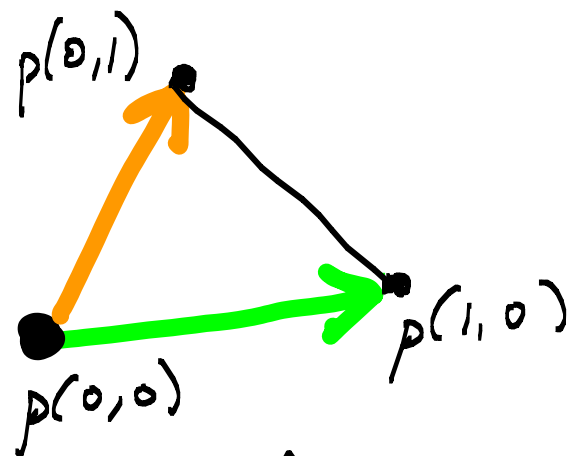
What are the sampling issues and how to deal with them ?

How to construct the homography ?



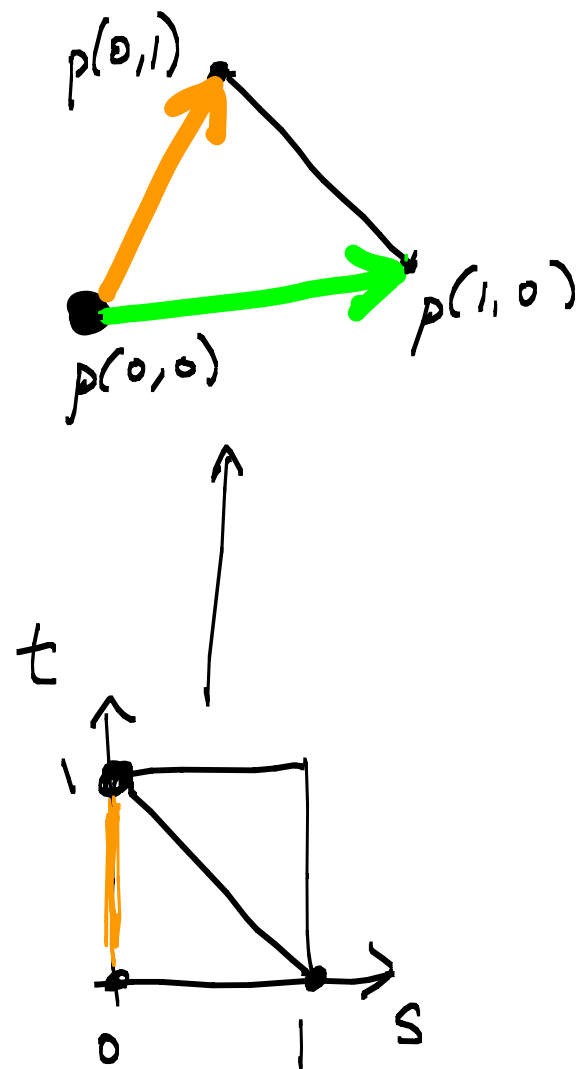
$$\vec{b} = p(0,1) - p(0,0)$$

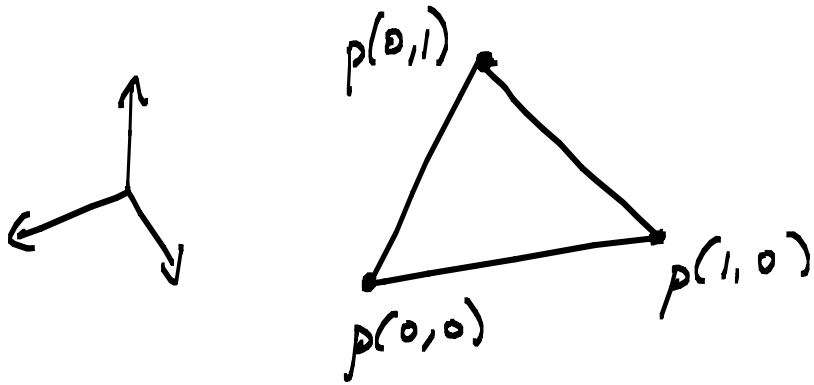
$$\vec{a} = p(1,0) - p(0,0)$$



$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \vec{a} & \vec{b} & p(0,0) \\ \hline 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ t \\ 1 \end{bmatrix}$$

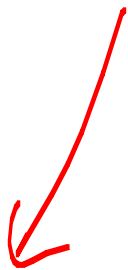
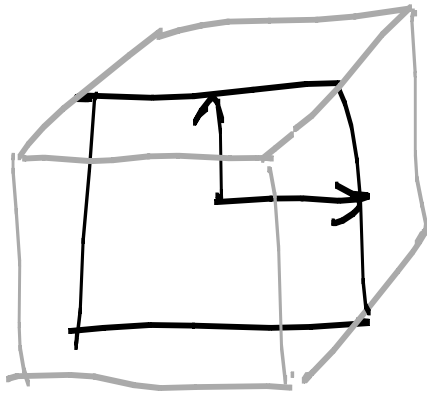
4×3





This is the usual projection from world coordinates to clip coordinates, but now the z row has been deleted.

3×4

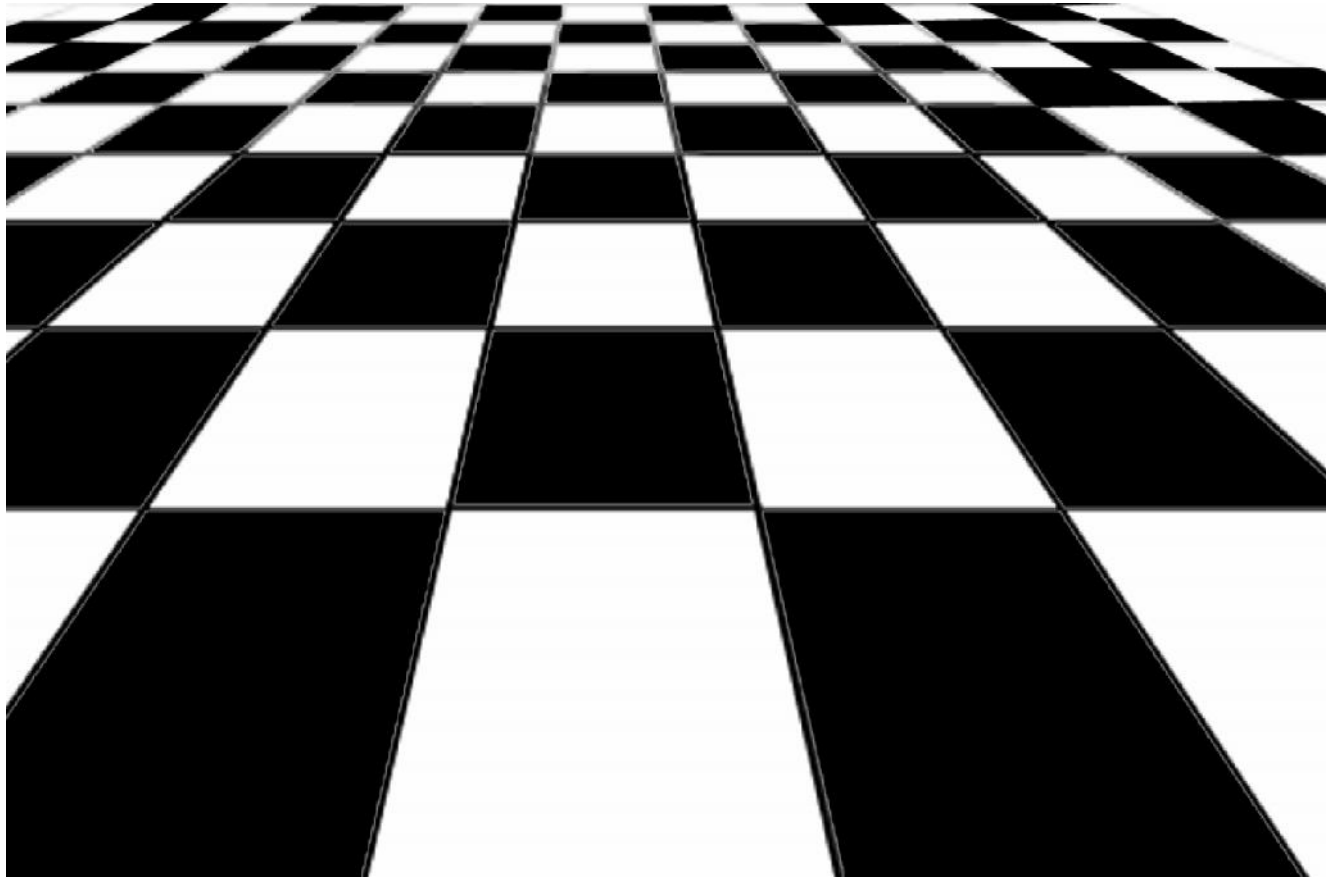



clip coordinates
(w x, wy, w)
with z deleted

Why? Because we can ignore hidden surface removal in this mapping.

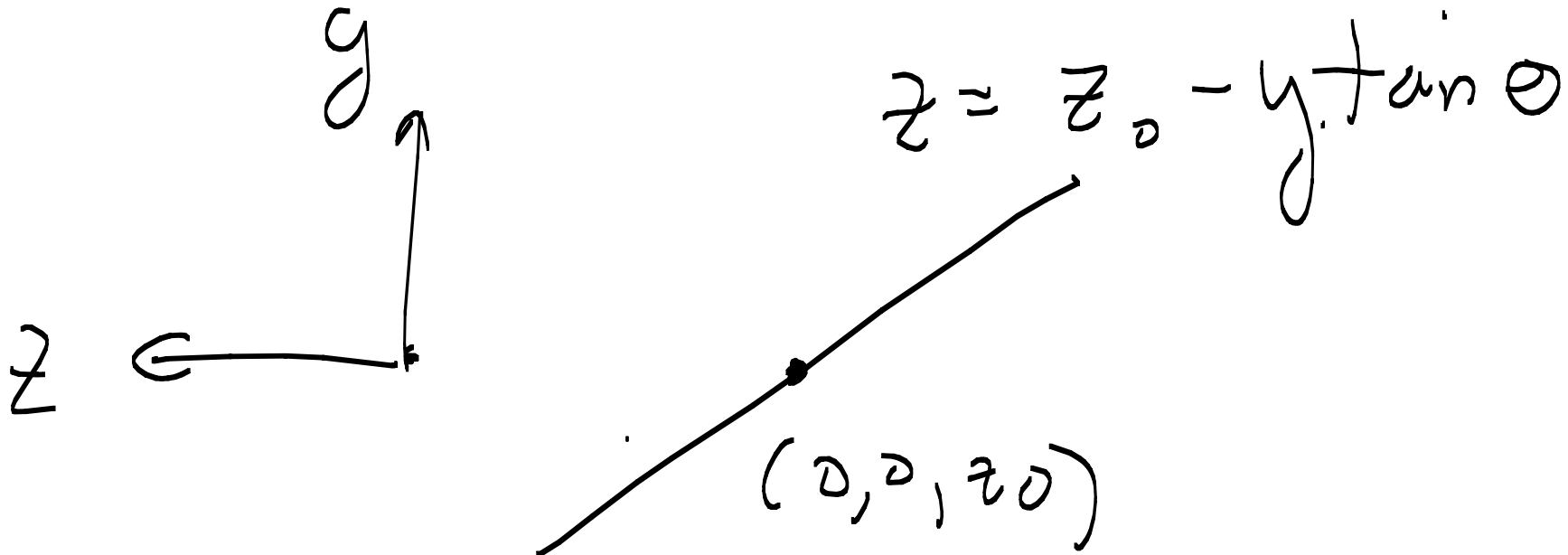
[ADDED April 25: I have ignored the normalization transformation in this example. This is only allowed in the special case that the normalization matrix is the identity matrix.]

Example: slanted floor



Example: slanted floor

Take $z = 0$ plane, rotate it by θ degrees around x axis, and translate it by z_0 .



Rotate by θ degrees around x axis, and translate by z_0 .

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

S
t
0
1

But we only apply this to points on the $z=0$ plane.

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & 0 \\ 0 & -\sin \theta & z_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ t \\ 1 \end{bmatrix}$$

3×4

4×3



$$\begin{bmatrix} \vec{a} & \vec{b} & p(0,0) \\ \hline 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ t \\ 1 \end{bmatrix}$$

Calculating H and its inverse gives....

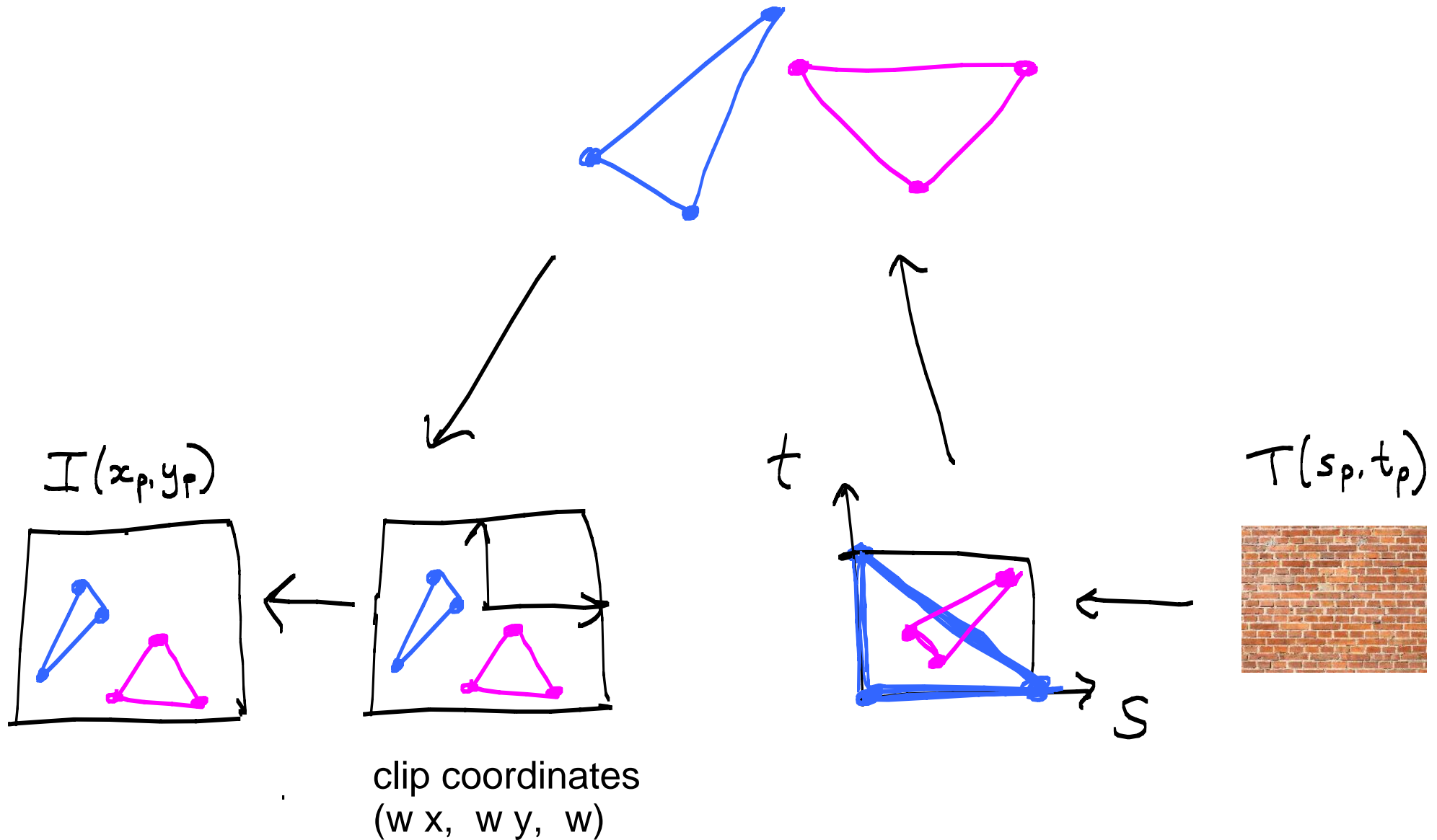
$$\mathbf{H} = \begin{bmatrix} f & 0 & 0 \\ 0 & f \cos \theta & 0 \\ 0 & -\sin \theta & z_0 \end{bmatrix}, \quad \mathbf{H}^{-1} = \begin{bmatrix} \frac{1}{f} & 0 & 0 \\ 0 & \frac{1}{f \cos \theta} & 0 \\ 0 & \frac{\tan \theta}{z_0 f} & \frac{1}{z_0} \end{bmatrix}$$

Exercise :

H maps which points in \mathbb{R}^2 to points at infinity (2D) ?

H maps points at infinity to which image points ?

OpenGL computes the homographies for you.



Exercise: where in the pipeline does this occur ?

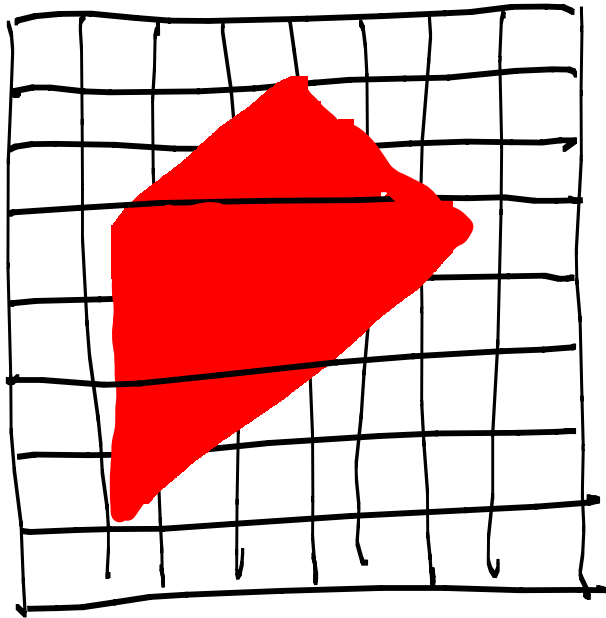
Details

How to construct the homography ?

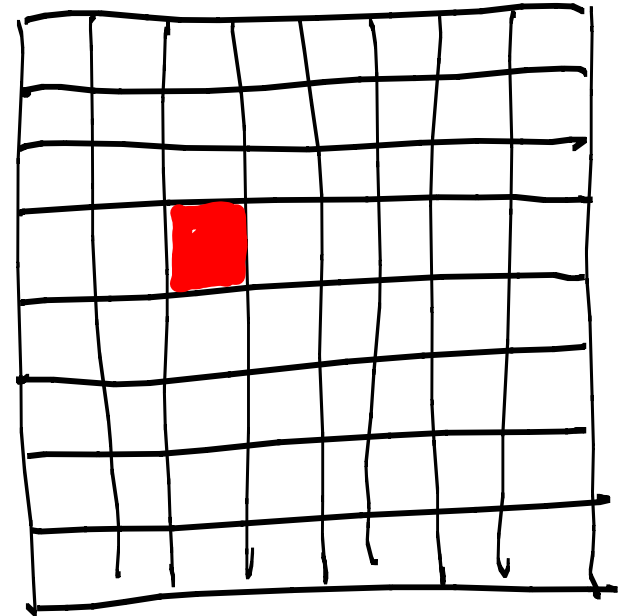
What are the sampling issues and how to deal with them ?

Texture magnification: a pixel in texture image ('texel') maps to an area larger than one pixel in image

$$I(x_p, y_p)$$

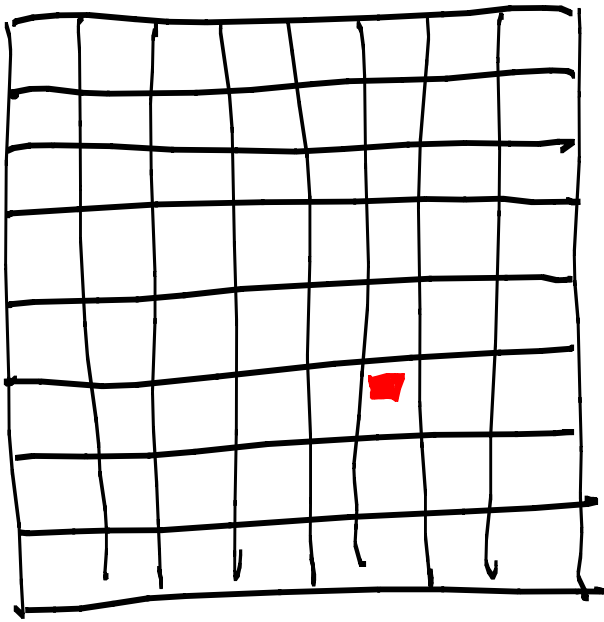


$$T(s_p, t_p)$$

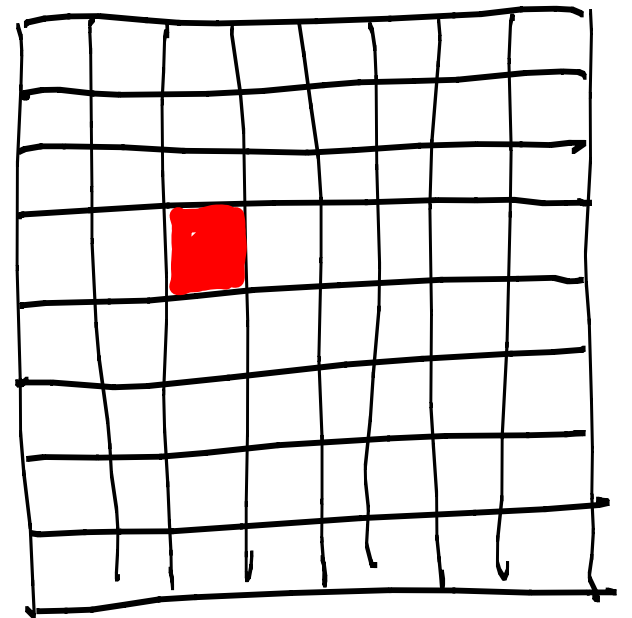


Texture minification: a pixel in texture image ('texel') maps to an area smaller than a pixel in image

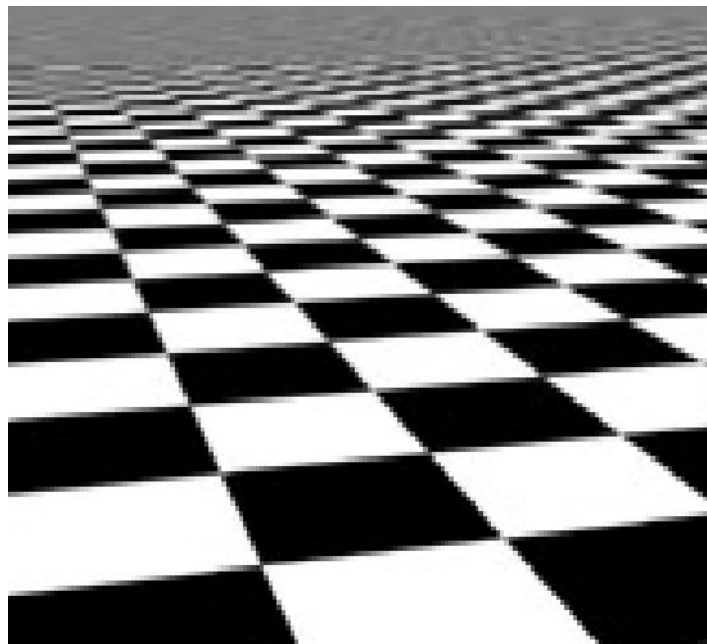
$$I(x_p, y_p)$$



$$T(s_p, t_p)$$

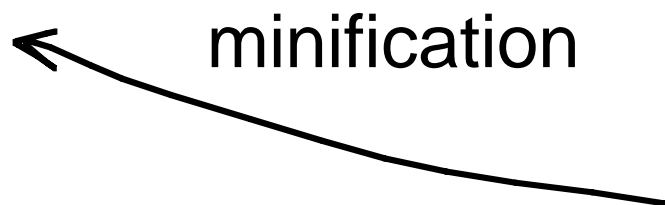


$$I(x_p, y_p)$$



e.g. 300 x 300

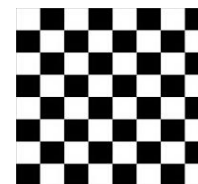
minification



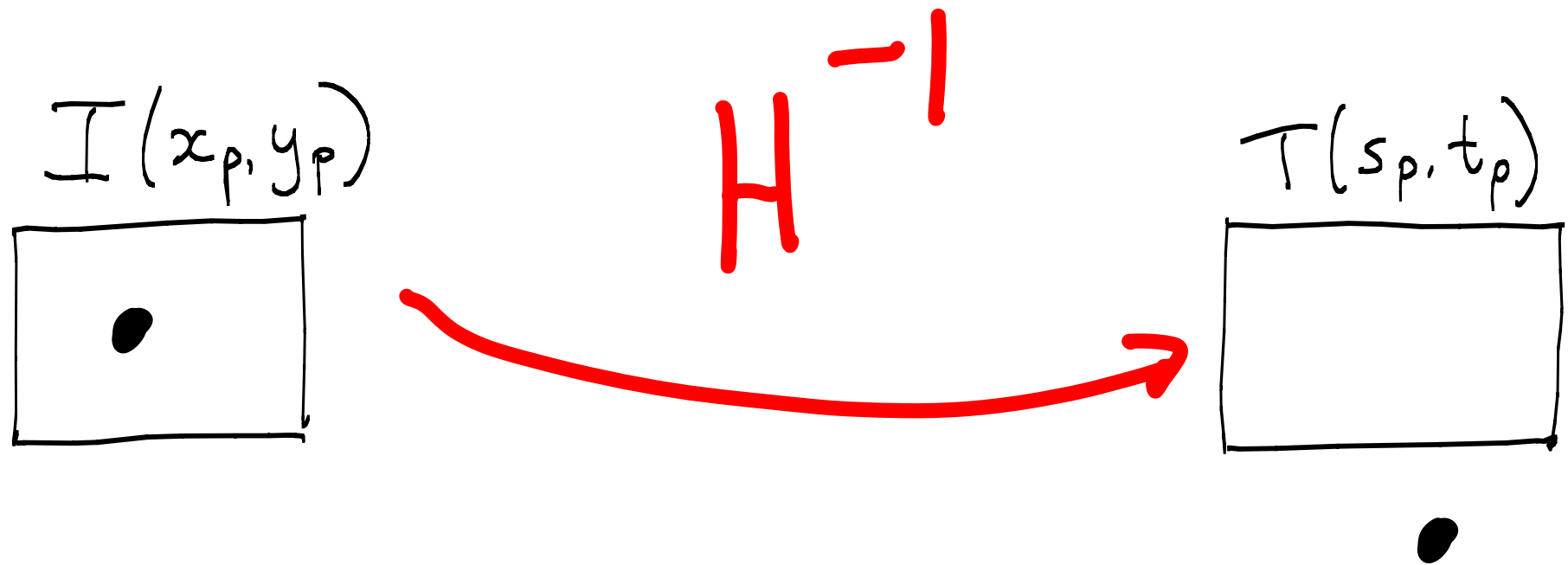
magnification



$$T(s_p, t_p)$$



e.g. 8 x 8



It can happen that inverse mapping is outside the range of the texture image. We need a policy in this case.
e.g. use $(x \bmod N_x, y \bmod N_y)$

Details

How to construct the homography ?

Q: What are the sampling issues ...

A: "Aliasing"

Q: ... and how to deal with them ?

A: "Anti-aliasing"

I will just give a sketch. A proper treatment would take several weeks.

"Aliasing" in computer graphics:

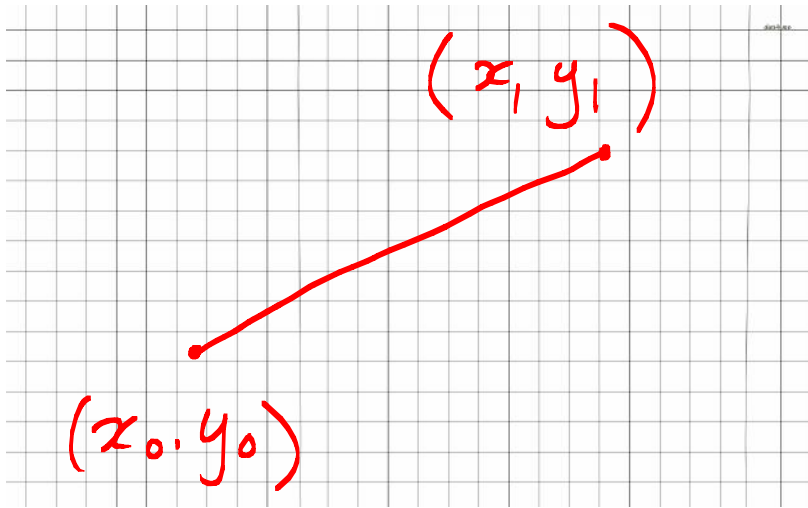
For any RGB image defined on a *discrete* grid of pixels, there are infinitely many images defined on the 2D *continuum*, that have the same RGB values at the discrete pixels.

"Aliasing" in programming languages:

Two variables reference the same memory location.

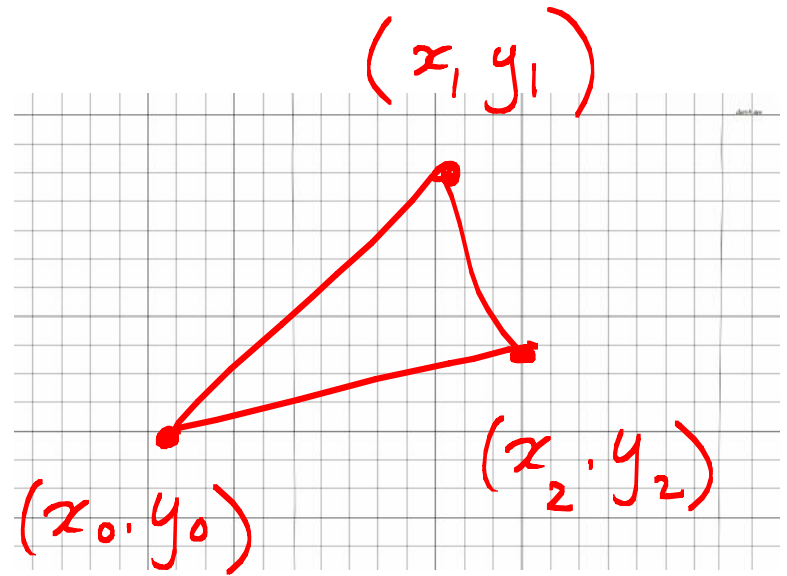
```
x = new Dog()  
y = x
```

Aliasing in scan conversion (lecture 6)



Line Segment

```
for x = round(x0) to round(x1) {  
    writepixel(x, Round(y) )  
    y = y + m  
}
```

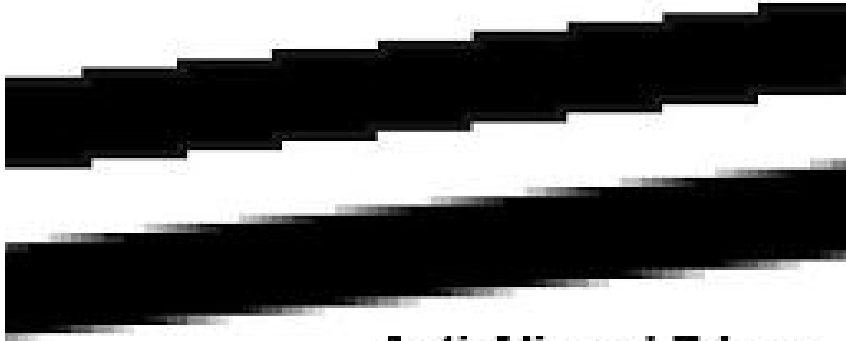


Polygon

```
for y = ymin to ymax {  
    compute intersection of polygon  
        edges with row y  
  
    fill in pixels between adjacent  
        pairs of edges  
}
```

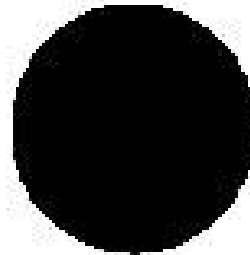
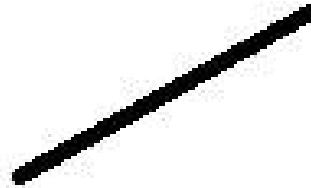
Aliasing and Anti-aliasing

Jagged Sharp Edges

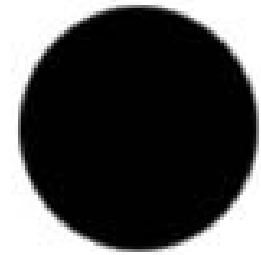


Anti-Aliased Edges

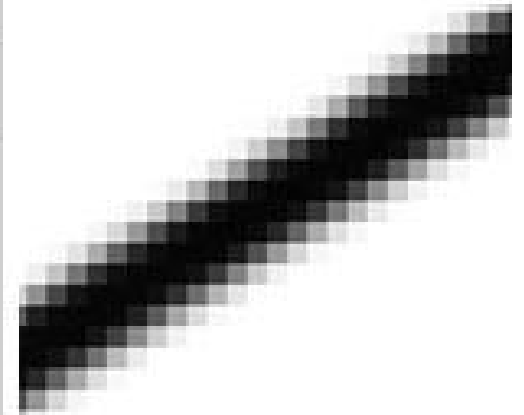
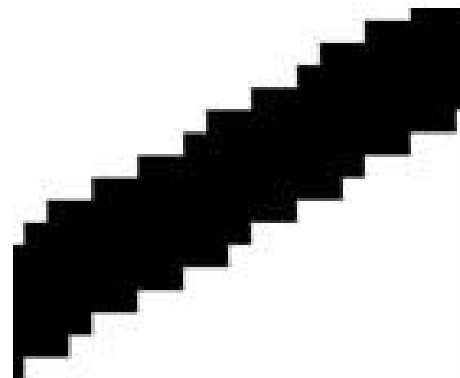
Aliased



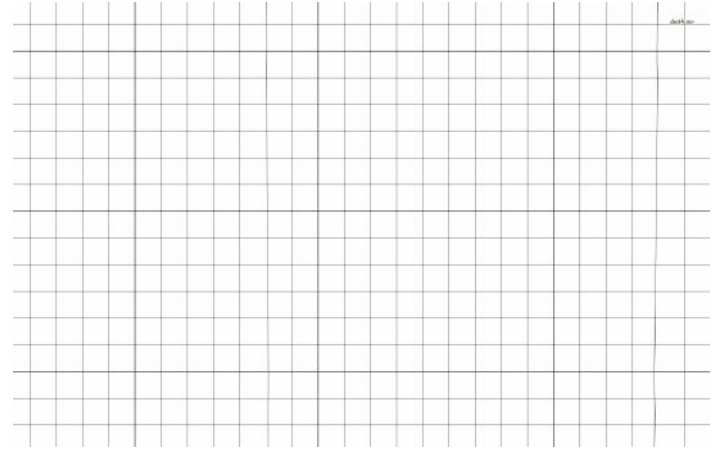
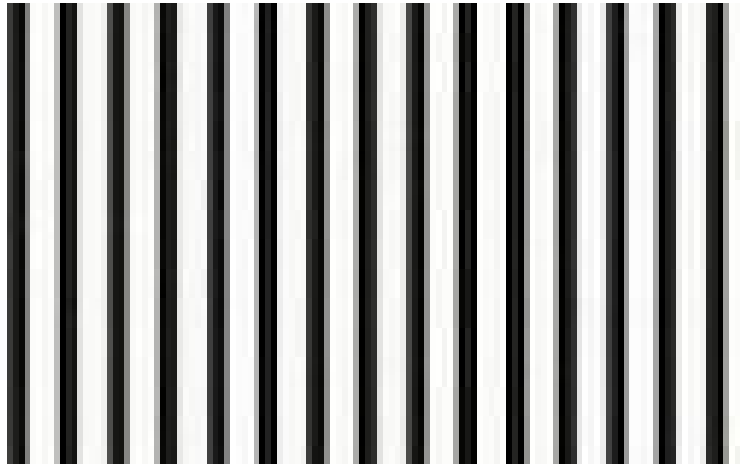
Anti-Aliased



note "big pixels" here
(magnification)



Textures ("regular") and aliasing



Suppose we sample the stripe image on the left using the intersection points (pixels) on the grid on the right.

Q: Will we also get *regular* vertical stripes in the sampled image?

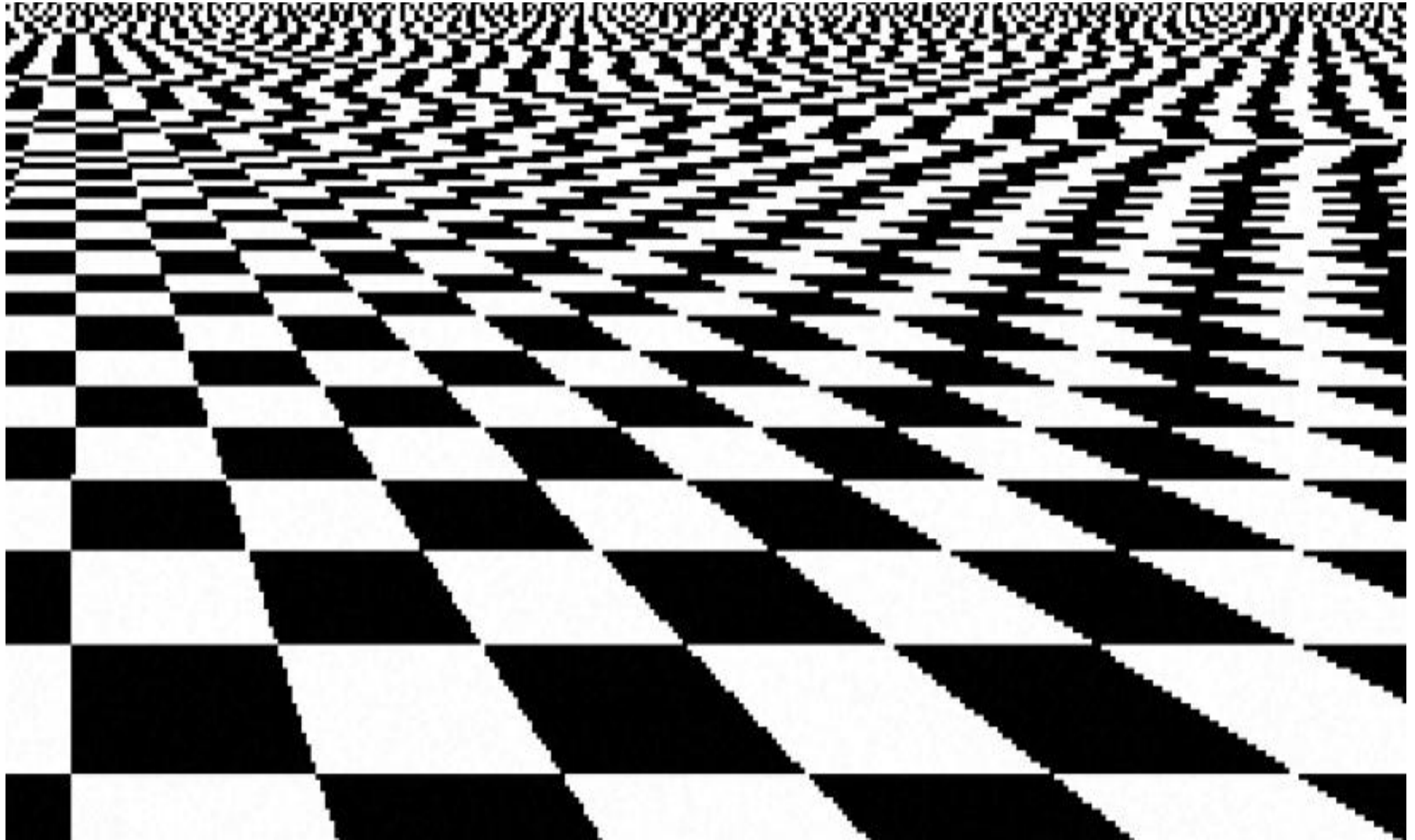
A: No, unless the distance between pixels happens to correspond exactly to the stripe width.

e.g. Moiré patterns



Caused by camera pixel frequency being higher than that of the grid pattern on the big central door (minification).

Somehow this slide was dropped from the lecture...
Too bad, because its a classic.

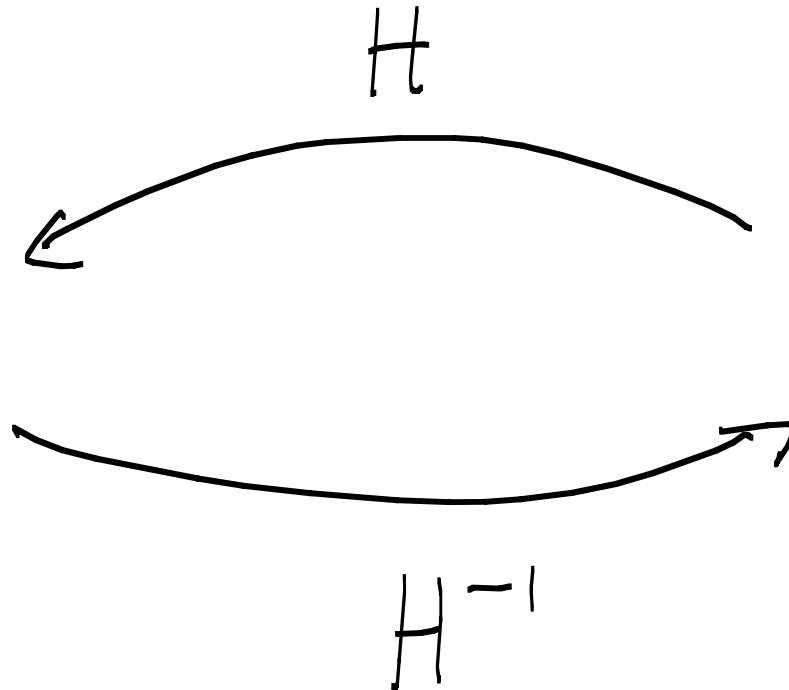
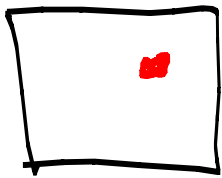


from Paul Heckbert, "Survey of Texture Mapping"

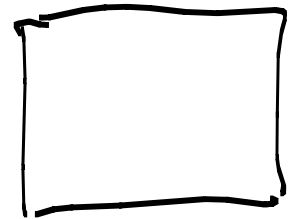
How to reduce aliasing in texture mapping ?

How to choose $I(x_p, y_p)$?

$I(x_p, y_p)$

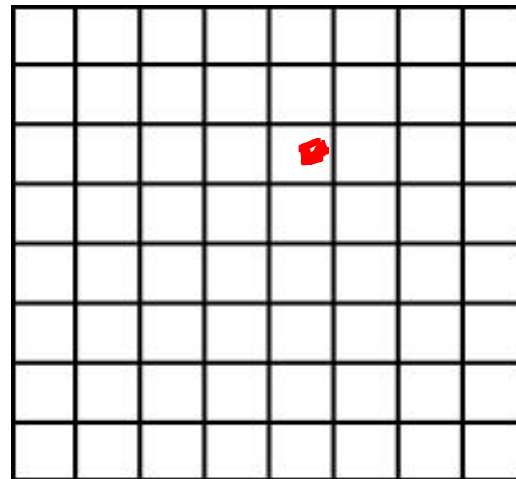
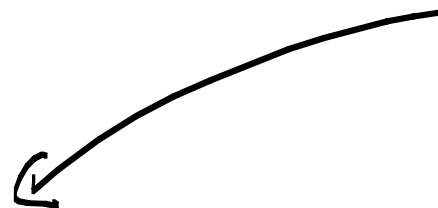
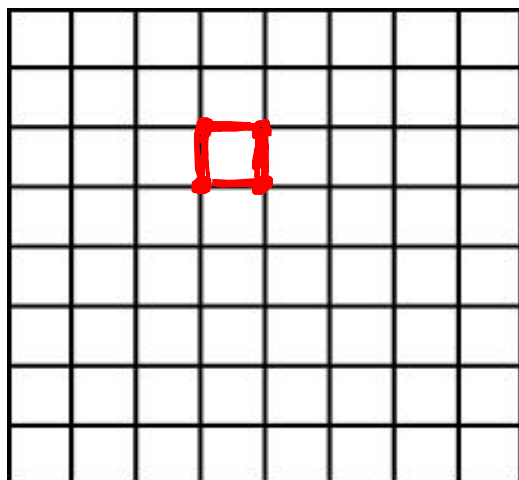


$T(s_p, t_p)$

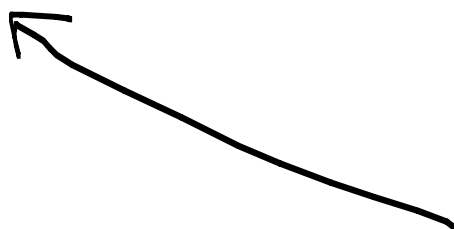


$$I(x_p, y_p)$$

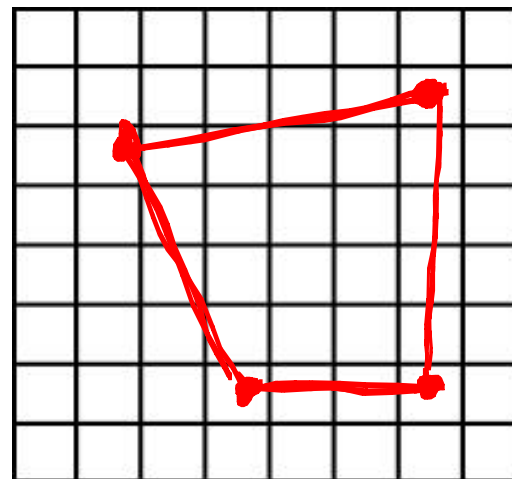
magnification



$$T(s_p, t_p)$$



minification

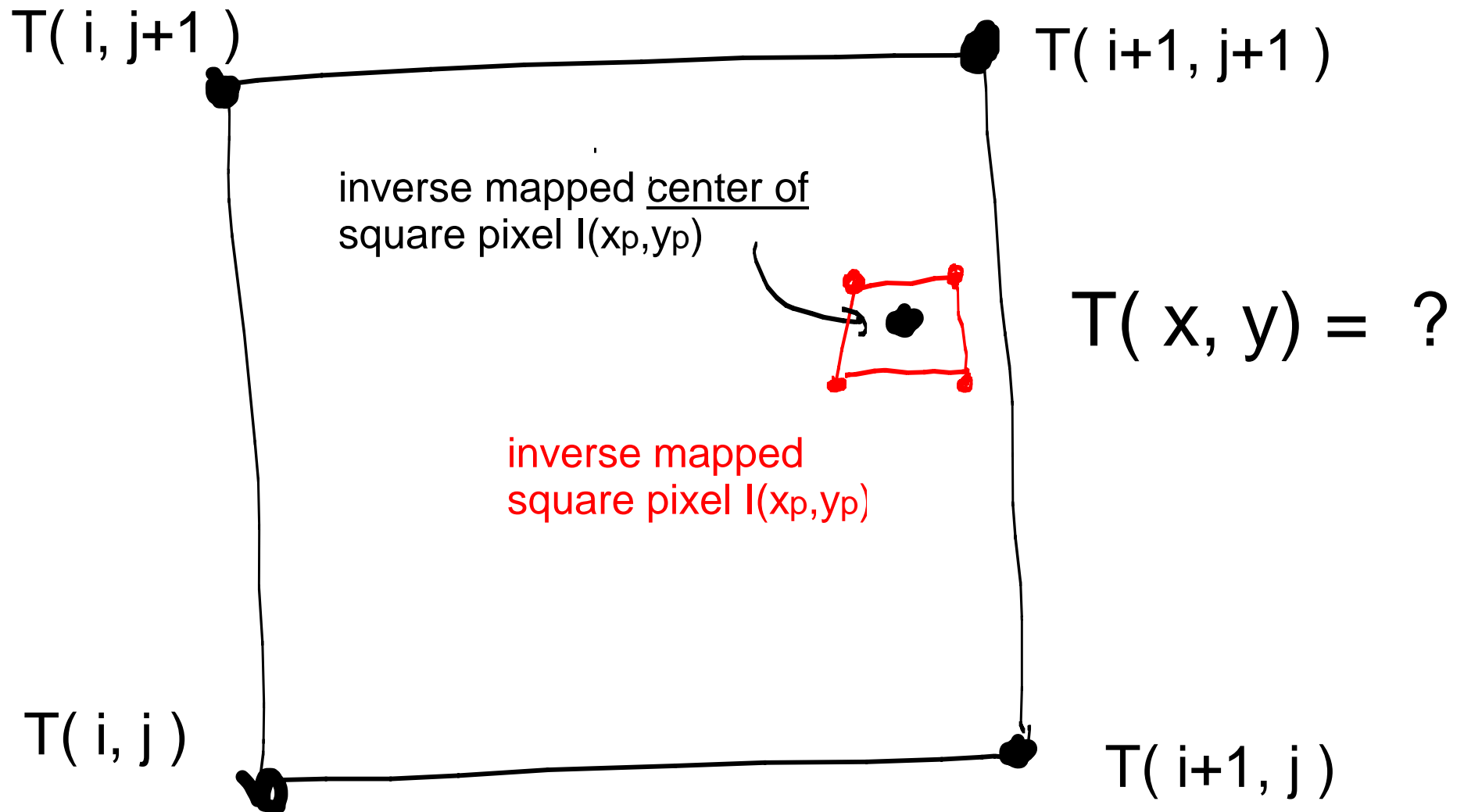


Change in notation for upcoming slides

Recall that a pixel can be thought of in two ways: as a little square, and as a point with integer coordinates. Texture mapping is a good example of why we need this flexibility. Up to now in this lecture, the pixels have been little squares. But in the following few slides, pixels *in the texture image* will be defined as a grid of points, namely the intersections of the horizontal and vertical lines of the grid. In particular, each square in the texture grid is no longer a pixel. Rather, the corner points of the square are the pixels.

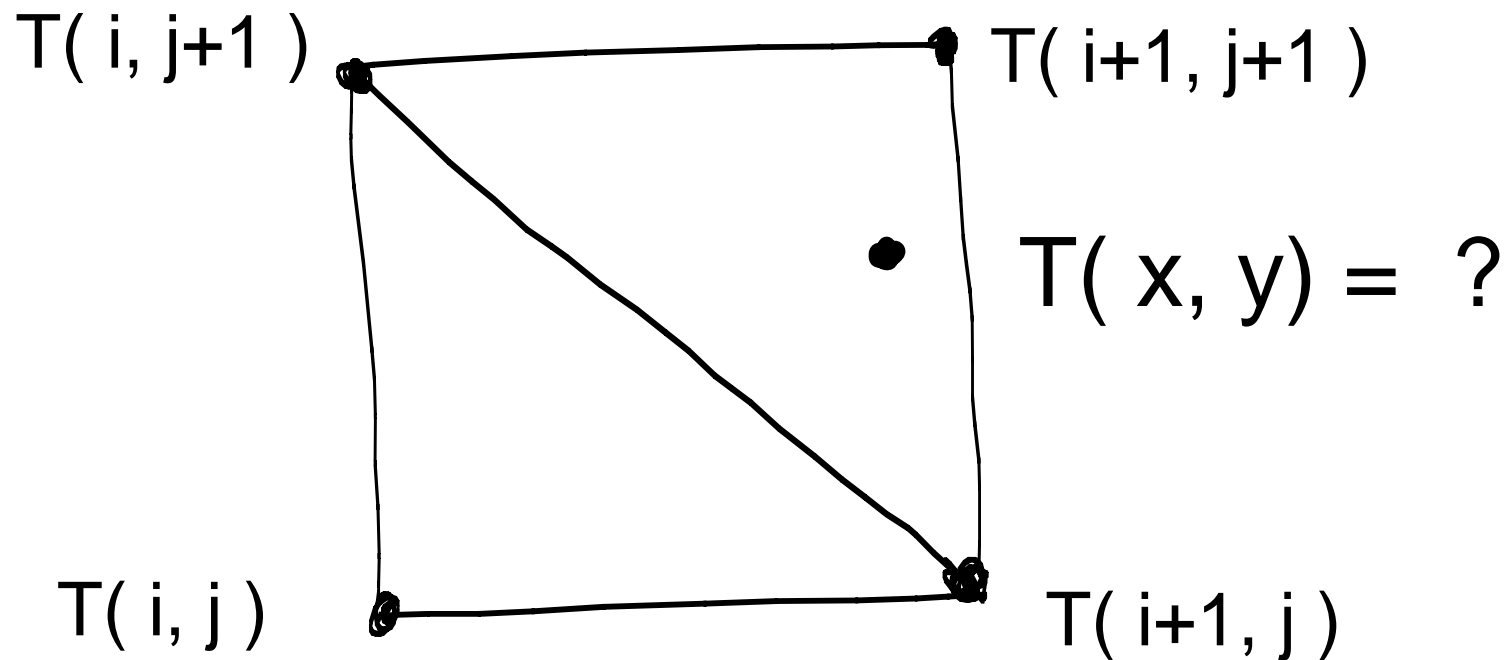
This should make more sense once you see the arguments.

Case 1: magnification



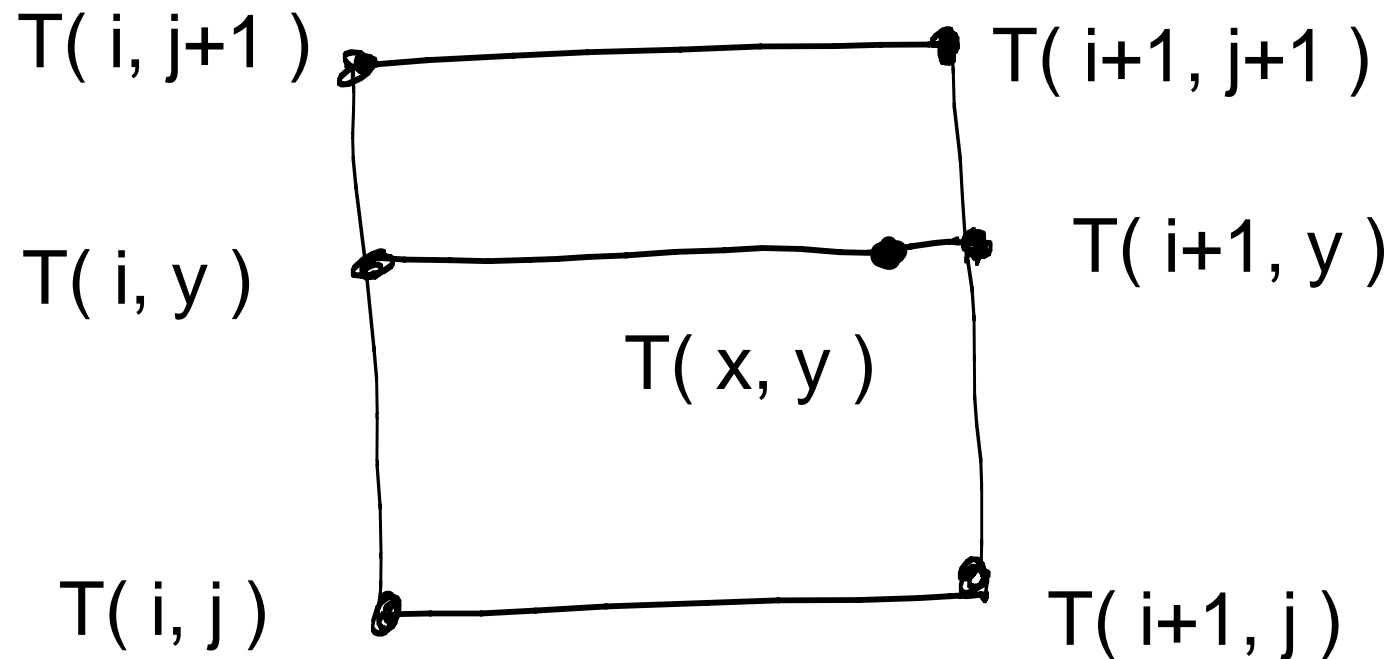
Solution 1a: Linear interpolation

Partition square into two triangles. Use linear interpolation within the triangle that (x,y) lies.



Exercise: What is the problem with this method ?

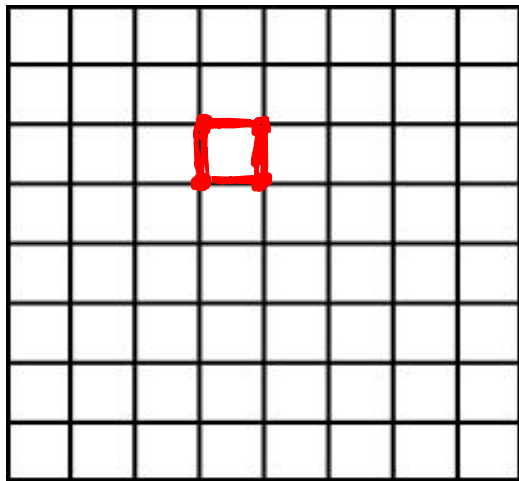
Solution 1b: "Bilinear interpolation"



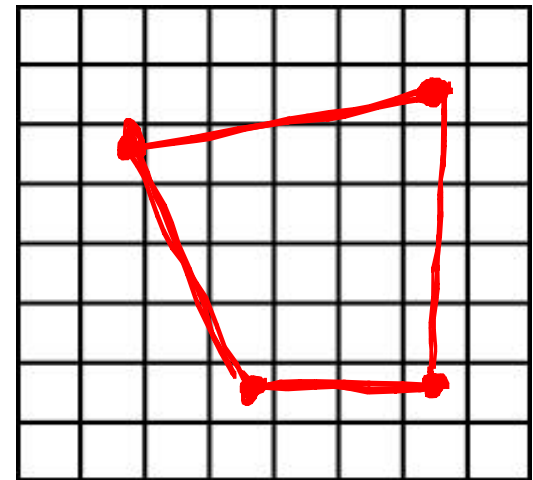
Exercise: Write out the formula for $T(x, y)$

Case 2: texture minification

$$I(x_p, y_p)$$



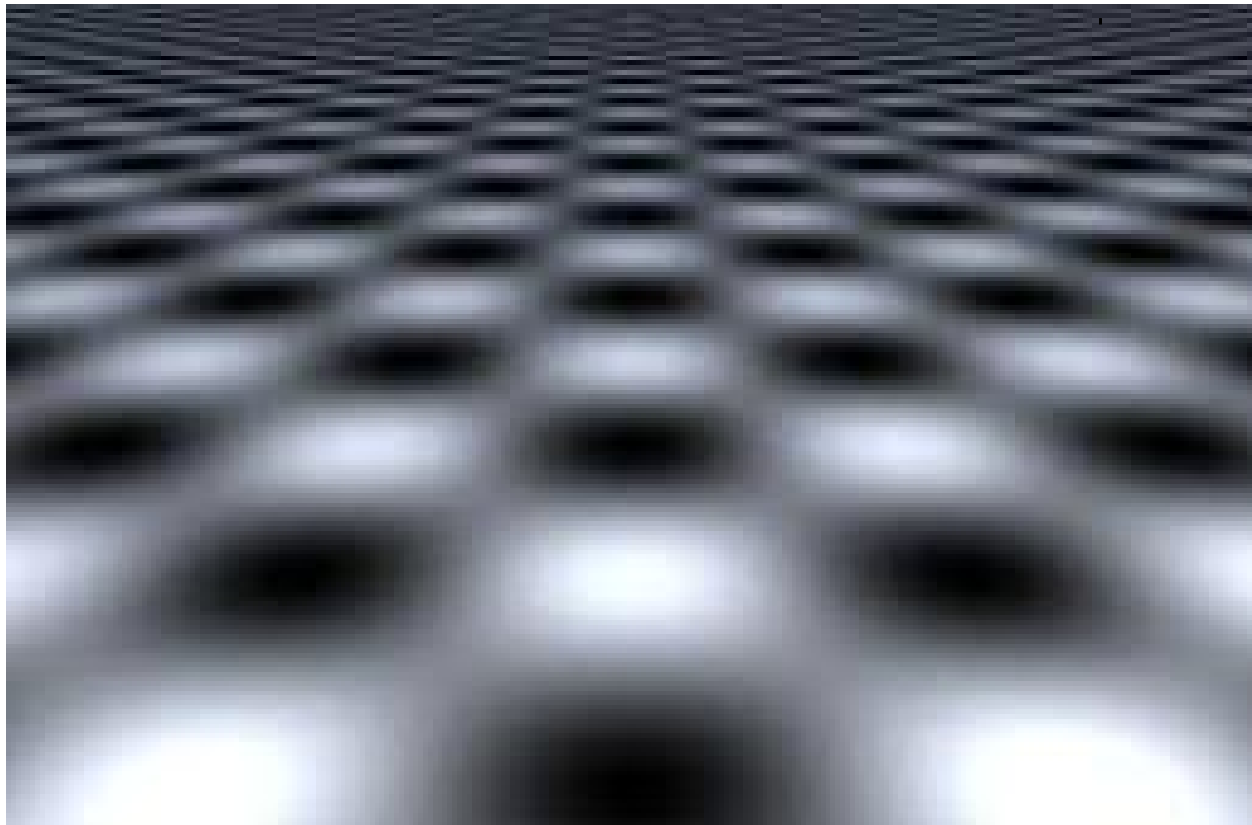
$$T(s_p, t_p)$$



Solution: (*not used by OpenGL*) Take average of intensities within the **quad (inverse map of square pixel)**.

OpenGL used MIP mapping (next lecture).

Here is an example of what these two solutions can produce. For minification, averaging produces grey pixels, which is appropriate. For magnification, the interpolation blurs the intensities, which is in this case doesn't work so well because there is too much of it. (We only wanted to blur the edge enough to hide the jaggies!)



minification

(solution:
averaging)

magnification

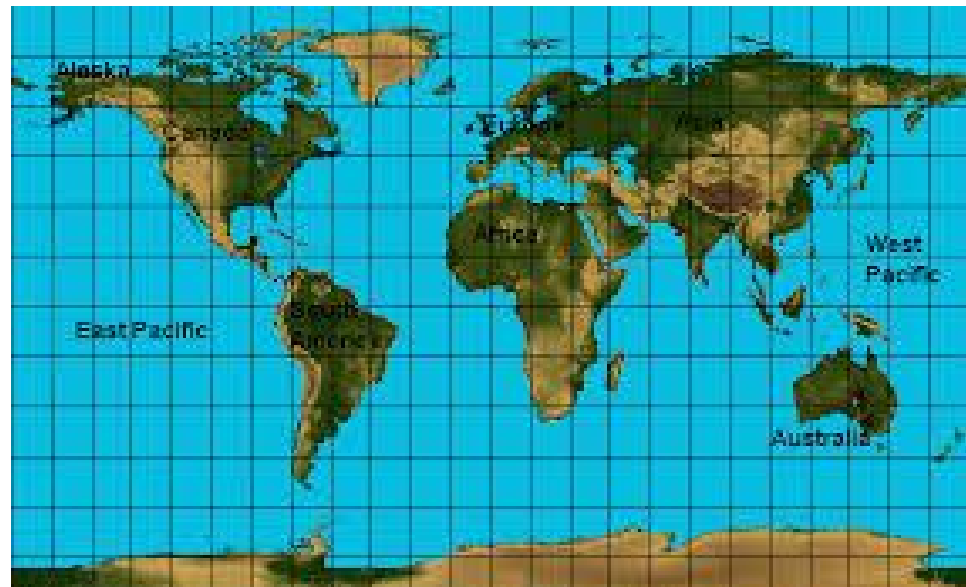
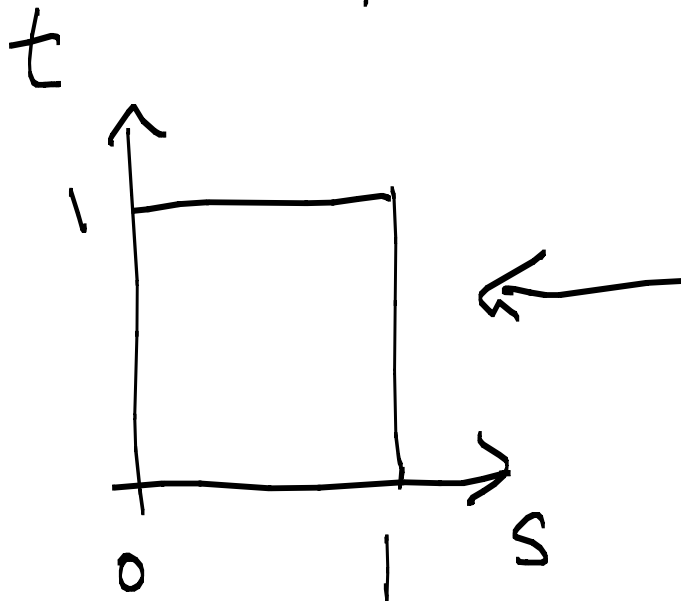
(solution:
interpolation)

How to texture map a quadric (or bicubic) ?



Discretize sphere into polygons (or use parametric surface patch).

$$T(s_p, t_p)$$



Texture mapping in OpenGL

(what is required?)

- a texture image

```
glTexImage2D( GL_TEXTURE_2D, ....., size, .., data )
```

- a correspondence between polygon vertices and texture coordinates

```
glBegin(GL_POLYGON)
  glTexCoord2f(0, 0);
  glVertex( ..... )
  glTexCoord2f(0, 1);
  glVertex( ..... )
  glTexCoord2f(1, 0);
  glVertex( ..... )
glEnd()
```