CSC 4356
Interactive Computer Graphics
Lecture 19: Texture Mapping (part 2)

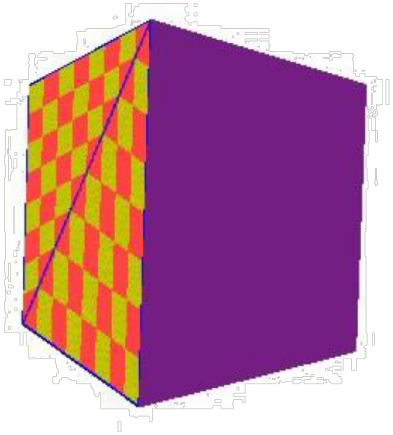
Jinwei Ye

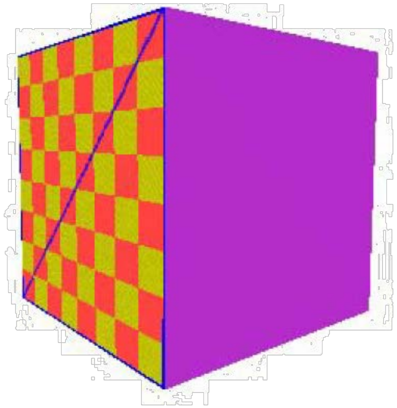
http://www.csc.lsu.edu/~jye/CSC4356/

Tue & Thu: 10:30 - 11:50am
218 Tureaud Hall

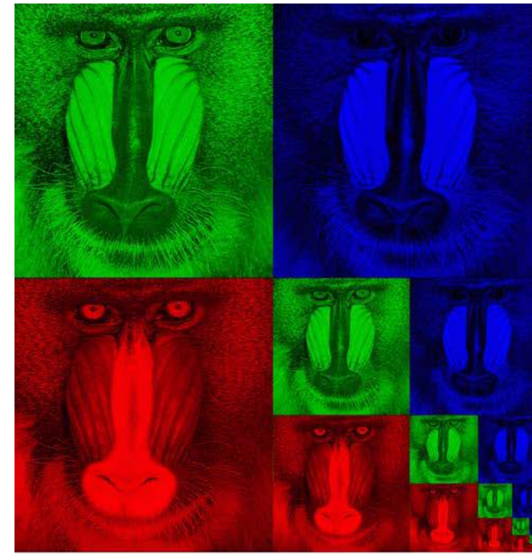
From Last Time

- Basic Texture Mapping
  - Planar, Cylindrical, Spherical, Cubic, etc.
- Texture Coordinate Interpolation
  - Linear interpolation is wrong (but fast)
  - Perspective-correct interpolation (interpolate in eye space instead of the screen space)





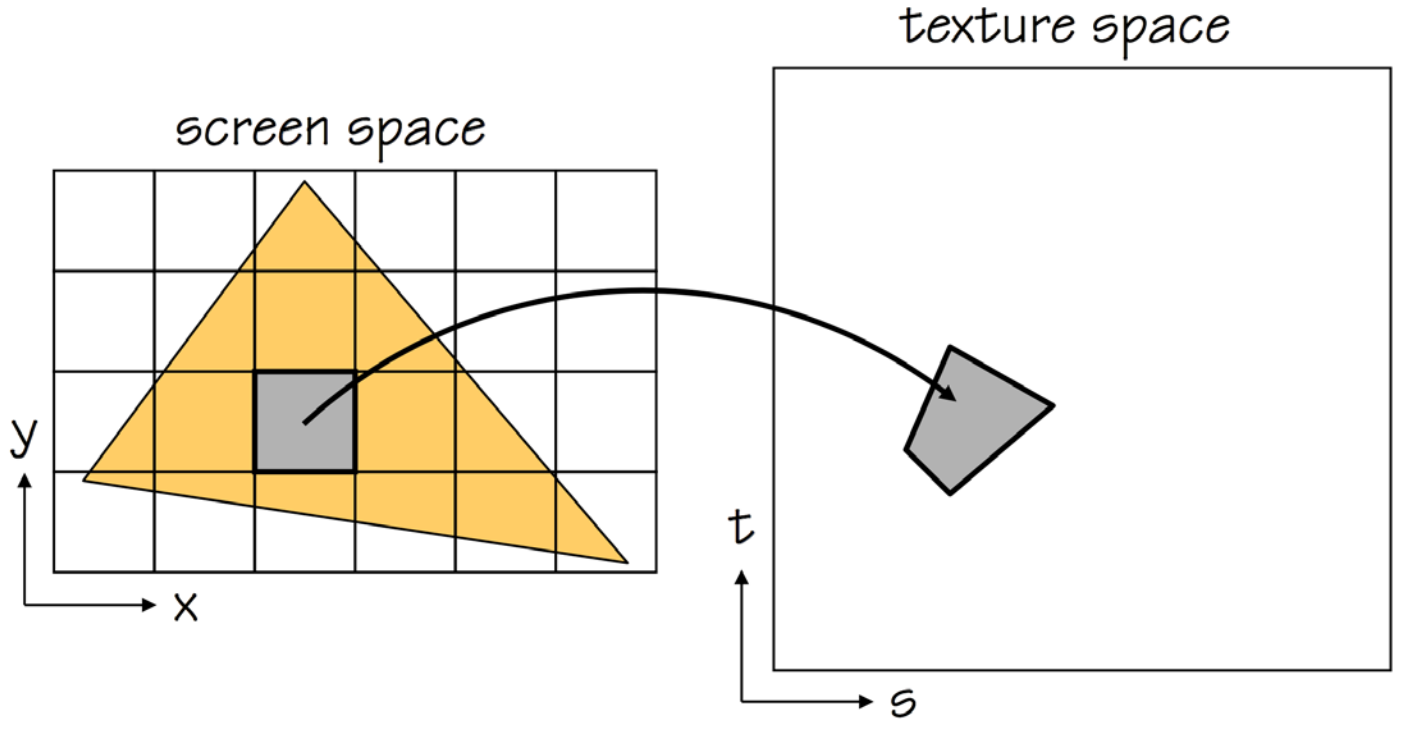
This Lecture



- Texture Filtering
  - Minification & Magnification
  - MIP Map
  - Summed-Area Table
- Texture Synthesis
  - How to create a large texture sample given a small one?

Sampling Texture Maps

- The uniform sampling pattern in screen space corresponds to some sampling pattern in texture space that is not necessarily uniform

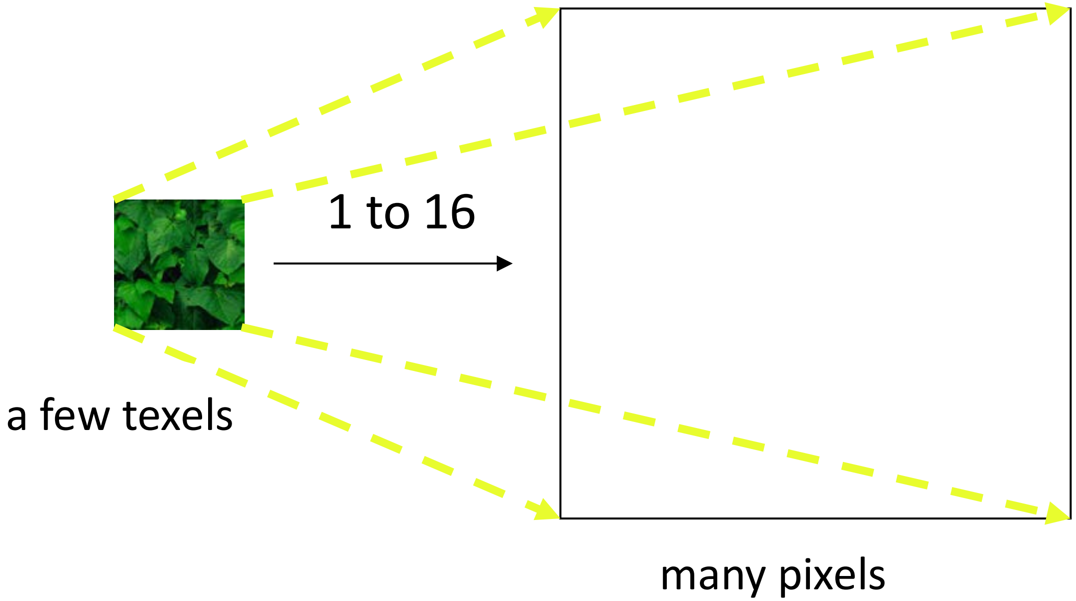


Sampling Density Mismatch

- Ideally, the mapping of texels to pixels would be one-to-one
- However, such mapping (sampling density) rarely matches
  - A small region of texels may be mapped to a large region of pixels (magnification)
  - A large region of texels may be mapped to a small region of pixels (minification)

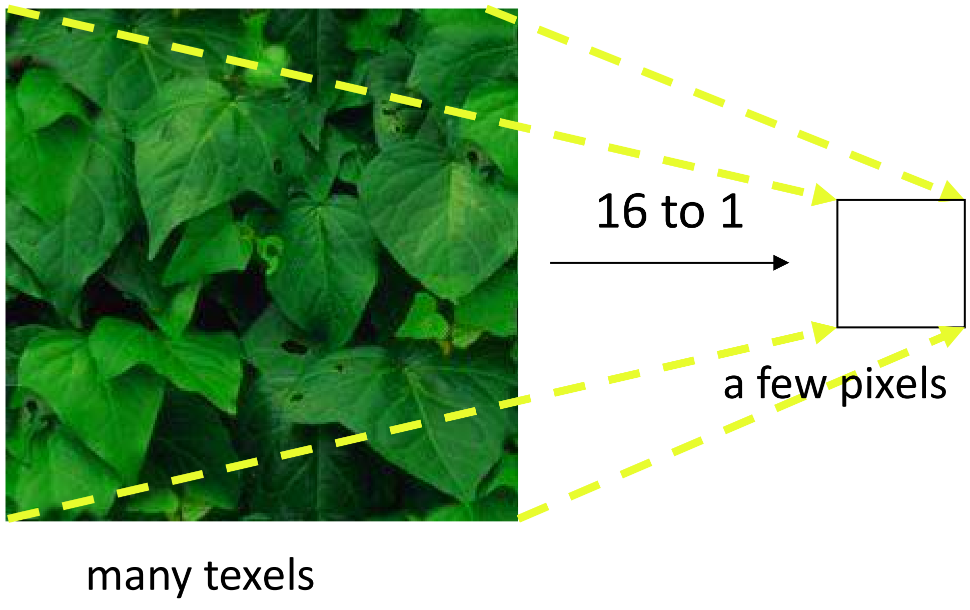
Magnification

- In magnification, one texel is mapped to many pixels

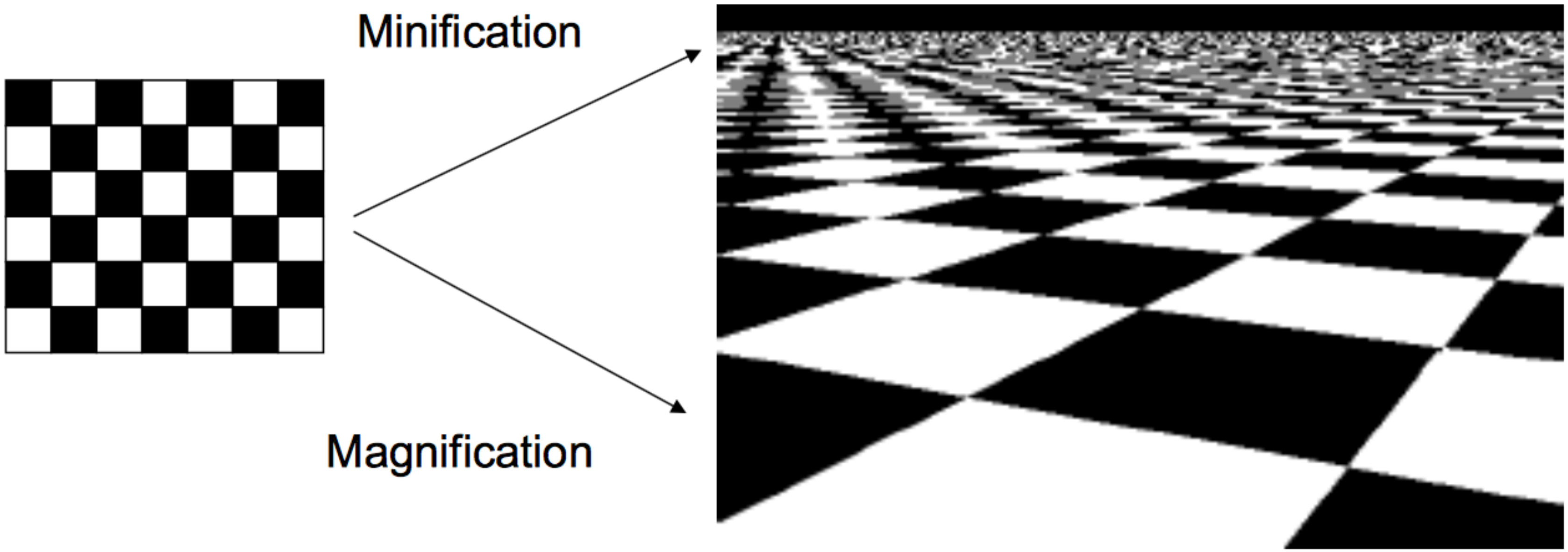


Minification

- In minification, many texels are mapped to one pixel



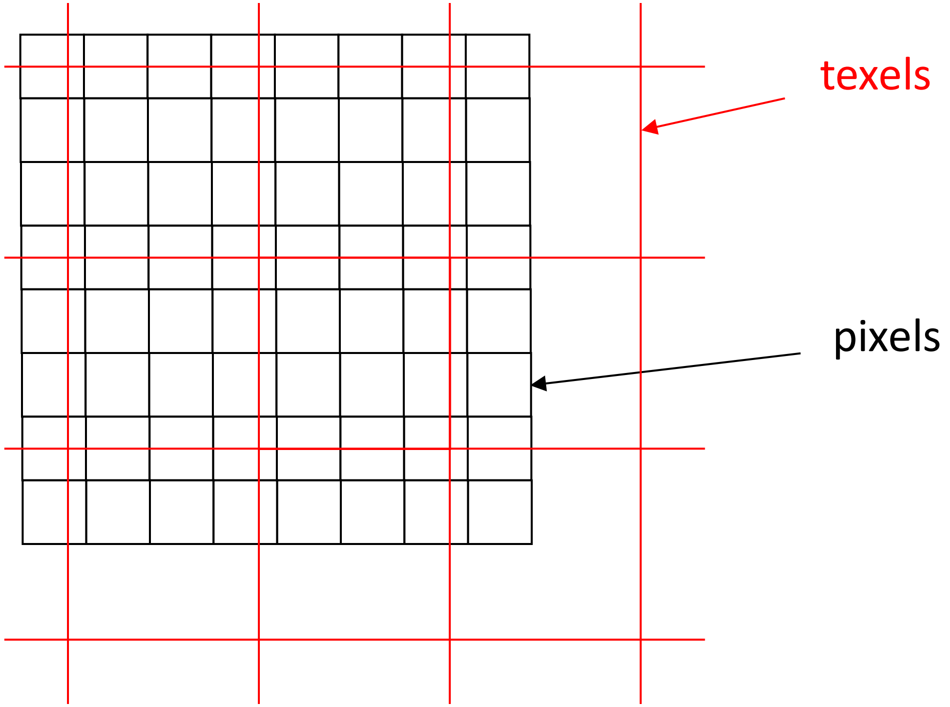
Minification & Magnification



(Note that minification is not handled very well here)

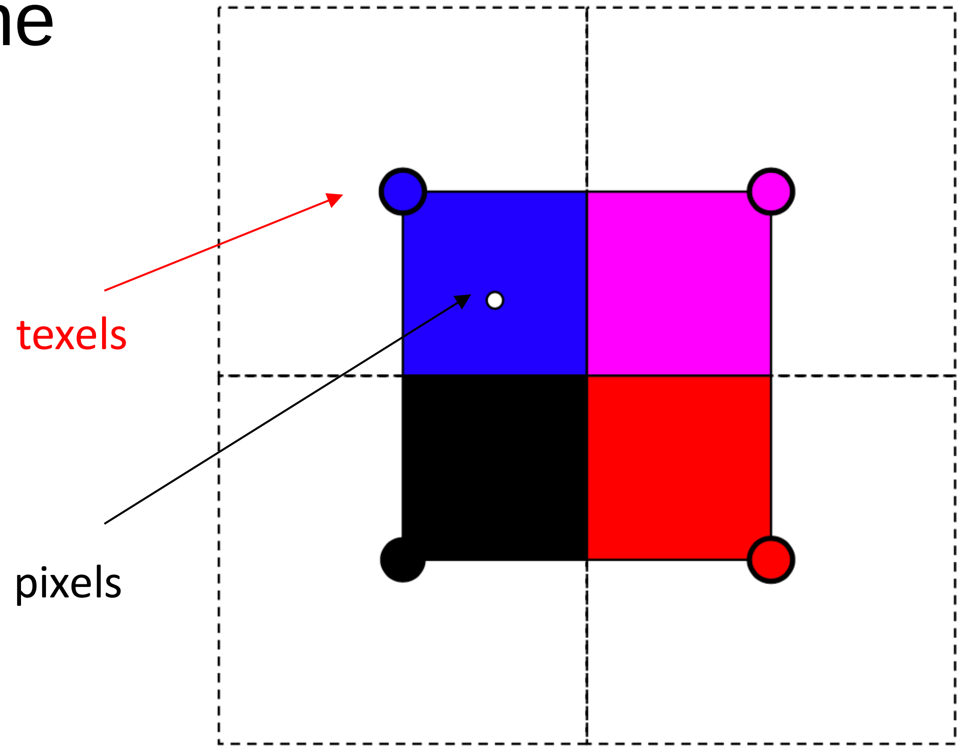
Handling Magnification

- Texels are larger than pixels
- How to compute color from texels to assign to a pixel?



Handling Magnification

- Nearest Neighbor
  - Take the color of the closest texel

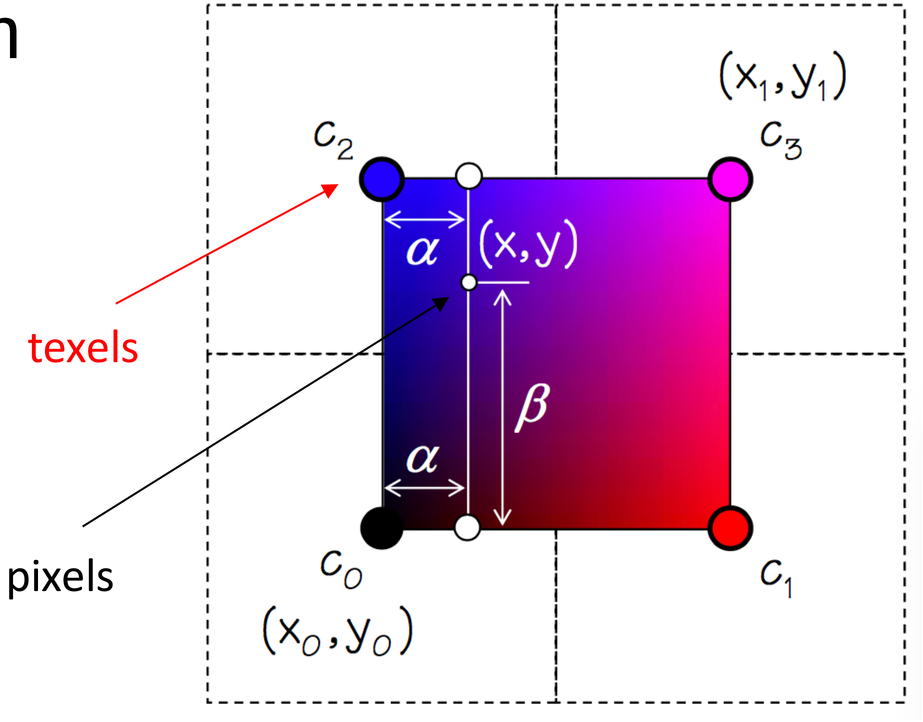


Handling Magnification

- Bilinear Interpolation
  - Interpolate color from four closest texels
  - Smooth appearance

$$\alpha = \frac{x - x_0}{x_1 - x_0} \qquad \beta = \frac{y - y_0}{y_1 - y_0}$$

$$c = ((1-\alpha)c_0 + \alpha c_1)(1-\beta) + ((1-\alpha)c_2 + \alpha c_3)\beta$$



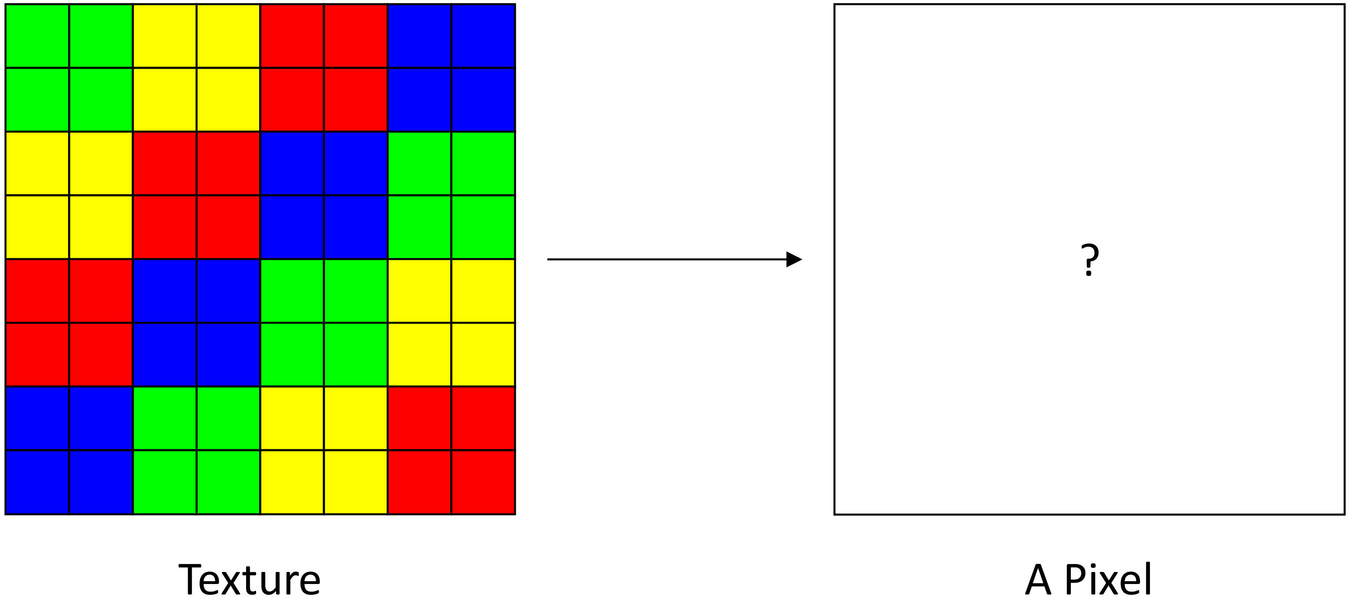
Handling Magnification

- Nearest neighbor vs. bilinear interpolation

Nearest neighbor bilinear interpolation

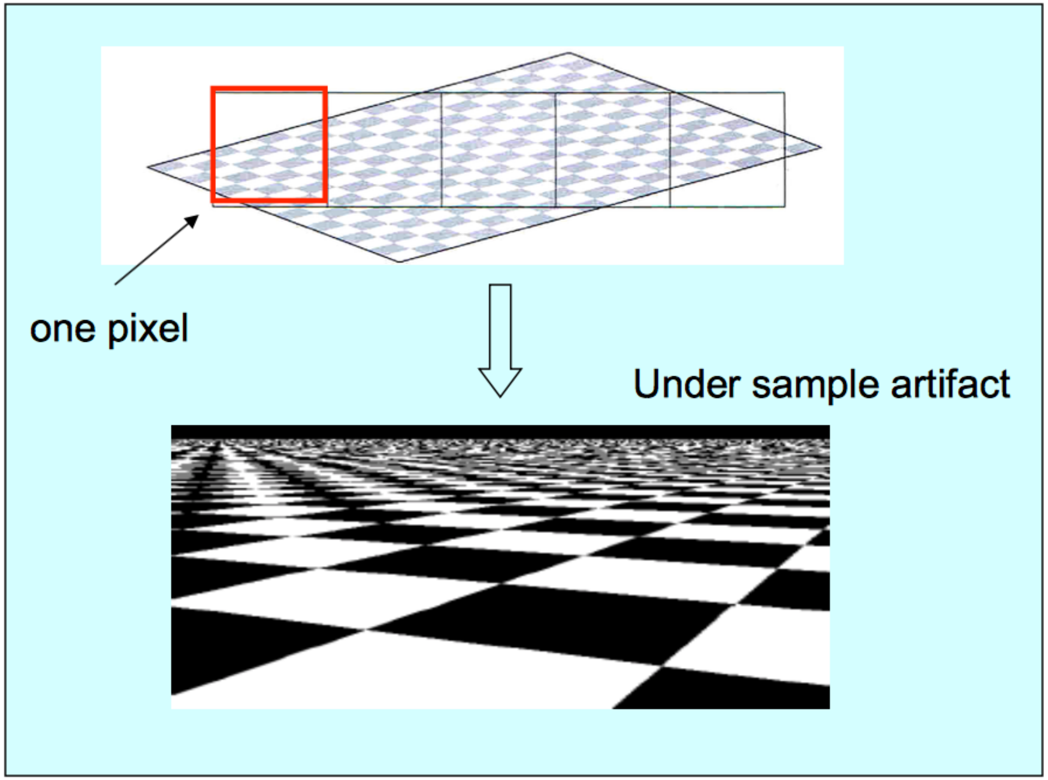
Minification

- Texels are smaller than pixels
  - Several texels covering one pixel
  - Multiple to one mapping
- How to compute color from texels to assign to a pixel?



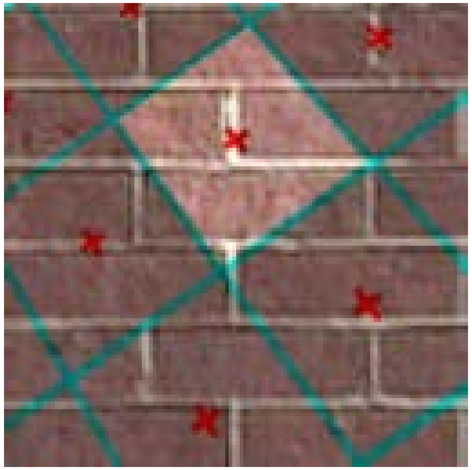
Aliasing Artifact

- Notice how details in the texture, in particular the mortar between the bricks, tend to pop (disappear and reappear). This popping is most noticeable around details (parts of the texture with a high-spatial frequency). This is called aliasing artifact.



Spatial Filtering

- To avoid aliasing, we need to pre-filter the texture to remove high frequencies
- Pre-filtering is essentially a spatial integration over the texture
- Texture integration on the fly is expensive
  - perform integration in a pre-process

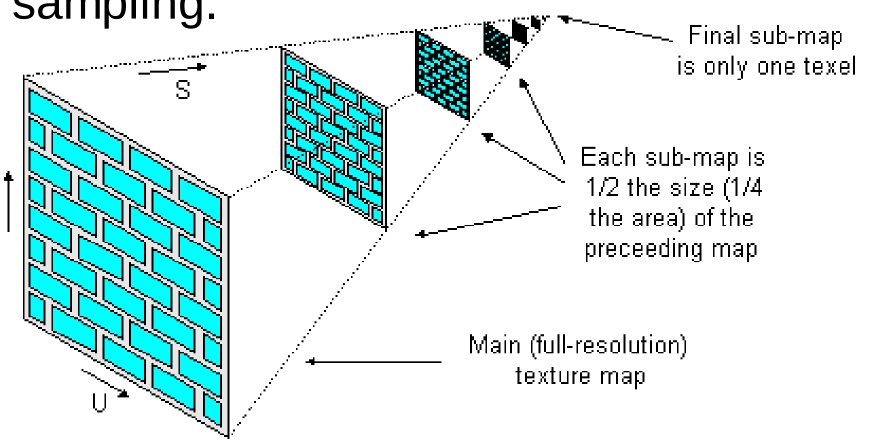


samples and their extents

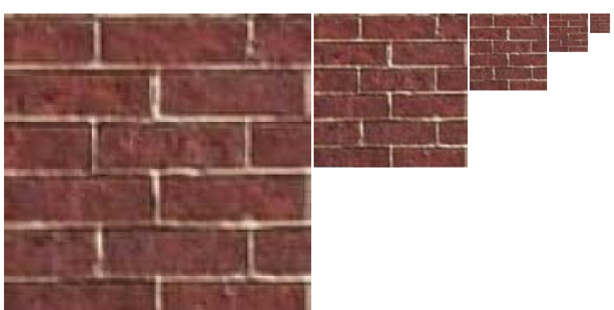
proper filtering removes aliasing

MIP Mapping

- MIP Mapping is one popular technique for precomputing and performing this pre-filtering. MIP is an acronym for the Latin phrase *multium in parvo*, which means "many in a small place". The technique was first described by Lance Williams. The basic idea is to construct a pyramid of images that are pre-filtered and resampled at sampling frequencies that are a binary fractions (1/2, 1/4, 1/8, etc) of the original image's sampling.



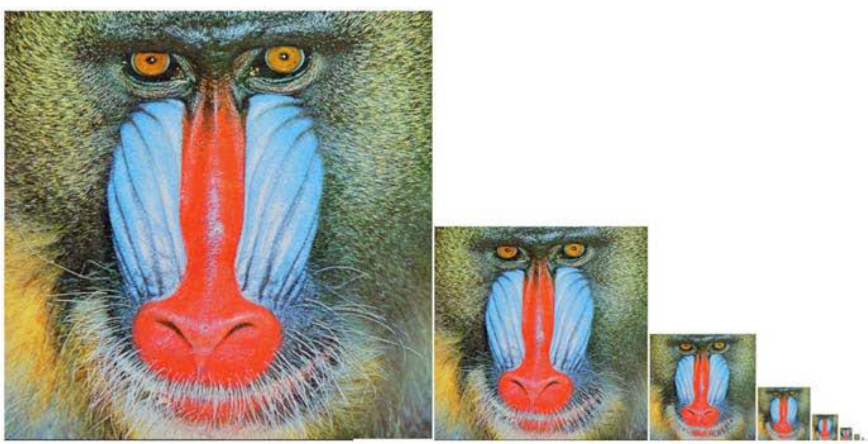
- While rasterizing we compute the index of the decimated image that is sampled at a rate closest to the density of our desired sampling rate
  - Try to maintain pixel to texel ratio close to 1



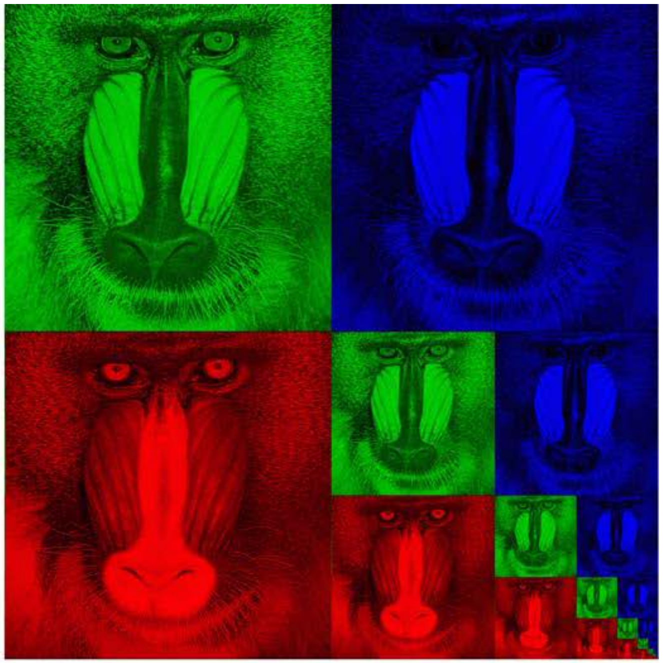
- Computing this series of filtered images requires only a small fraction of additional storage over the original texture (How small of a fraction?)

Storing MIP Maps

- One convenient method of storing a MIP map is shown on the right image
  - It also nicely illustrates the 1/3 overhead of maintaining the MIP map



10-level mip map



Memory format of a mip map

$$mip\ map\ size = \sum_{i=0}^{\infty} \left(\frac{1}{4}\right)^i = \frac{1}{1 - \frac{1}{4}} = \frac{4}{3}$$

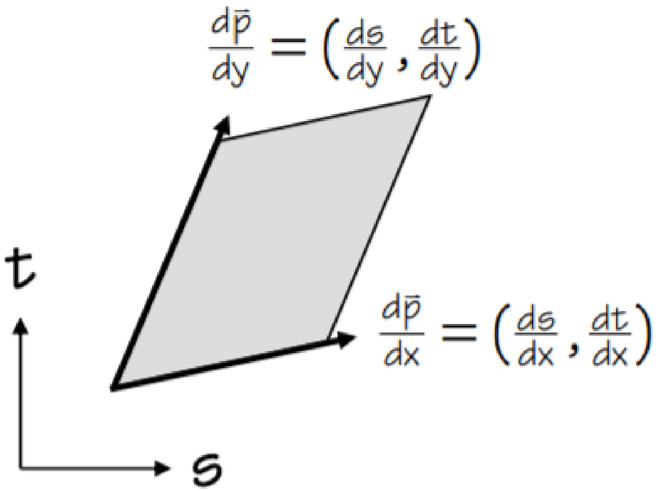
Finding MIP Level

- Idea: Use the projection of a pixel in screen into texture space to figure out which level to use

$$u^*(x,y) = u/w = A_u x + B_u y + C_u$$

$$o^*(x,y) = 1/w = A_o x + B_o y + C_o$$

$$s = u \cdot textureWidth$$



Applying chain rule: $\frac{ds}{dx} = \frac{ds}{du}\frac{du}{dx}$

$$\frac{ds}{du} = textureWidth$$

$$\frac{du}{dx} = \frac{d(u^*(x,y)/o^*(x,y))}{dx} = \frac{A_u o^*(x,y) - A_o u^*(x,y)}{o^*(x,y)^2}$$

Other derivatives can be computed in the same way.

Finding MIP Level

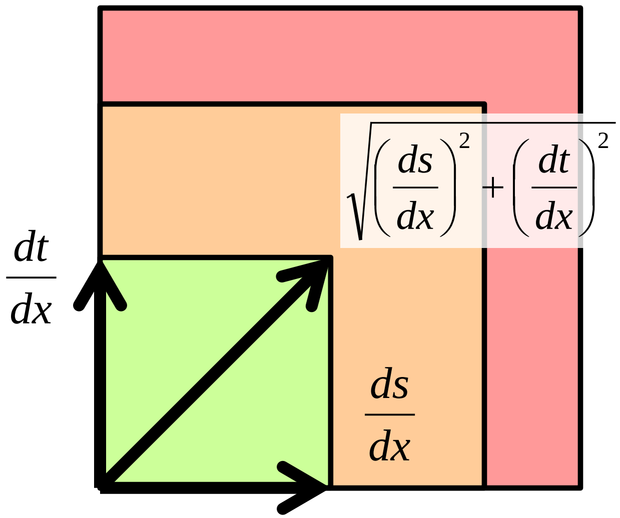
- Use the lengths of the projected pixel in texture space to define a measure of mismatch between sampling densities:

$$m = \max\left(\left\|\frac{d\vec{p}}{dx}\right\|, \left\|\frac{d\vec{p}}{dy}\right\|\right) = \max\left(\sqrt{\left(\frac{ds}{dx}\right)^2 + \left(\frac{dt}{dx}\right)^2}, \sqrt{\left(\frac{ds}{dy}\right)^2 + \left(\frac{dt}{dy}\right)^2}\right)$$

$$\approx \max\left(\max\left(\frac{ds}{dx}, \frac{dt}{dx}\right), \max\left(\frac{ds}{dy}, \frac{dt}{dy}\right)\right)$$

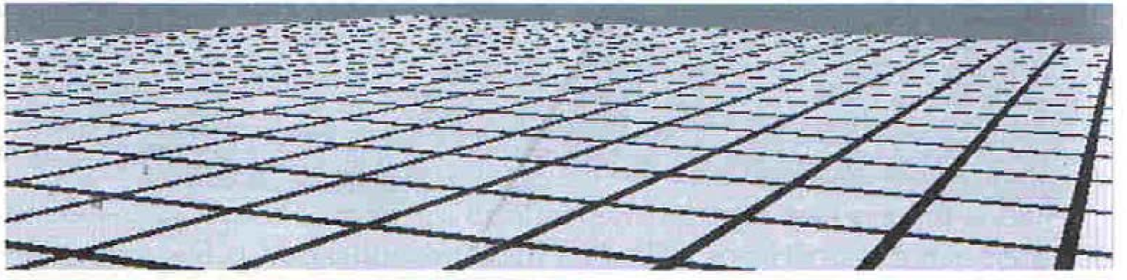
- Now choose the appropriate level:

$$level = \left\lfloor \log_2(m) \right\rfloor$$

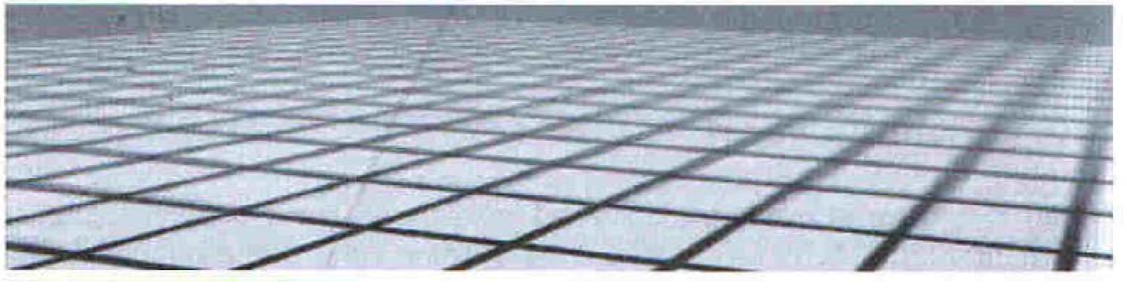


MIP Mapping Problem

- Overblurring!
  – Isotropic filtering
  – When a pixel covers many u texels but few v texels, we always choose the largest pixel coverage to decide the level



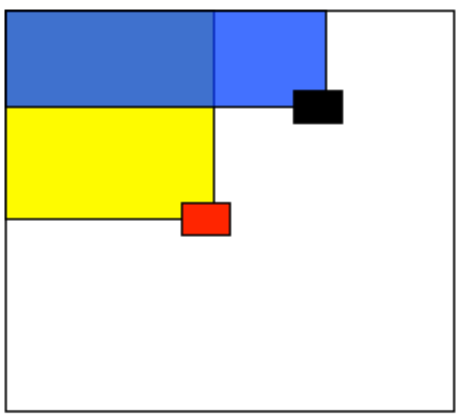
Non-antialiasing



MIP mapping

Summed-Area Tables

- Summed-area tables perform anisotropic filtering
  - It can be used to compute the average color for any arbitrary rectangular region in the texture space at a constant speed
- Summed-area table is a two dimensional array that has the same size as the texture



Each entry stores the sum of all the texel colors above and to the left

Summed-Area Tables

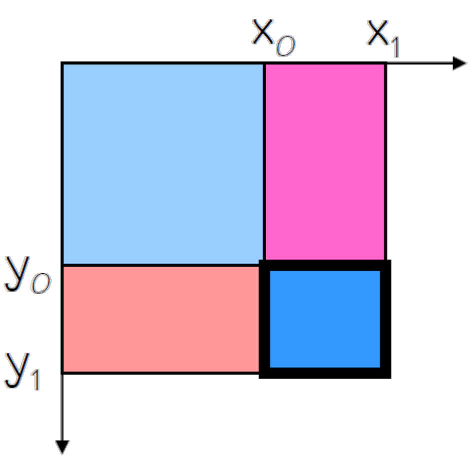
- Each entry in the summed area table is the sum of all entries above and to the left:

1	6	8	3
0	0	3	7
4	7	8	8
5	0	9	9

Original Texture

1	7	15	18
1	7	18	28
5	18	37	55
10	23	51	78

Summed-Area Table



- How to compute the color of a pixel bounded by (x0, y0) and (x1, y1)?
  - Find the sum of region contained in a box bounded by (x0, y0) and (x1, y1):

$$T(x_1, y_1) - T(x_0, y_1) - T(x_1, y_0) + T(x_0, y_0)$$

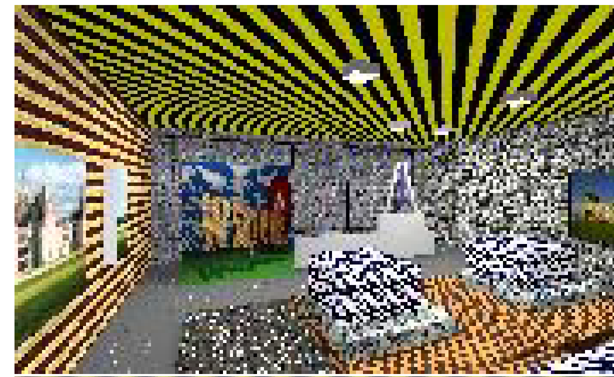
  - Divide out area

$$(y_1 - y_0)(x_1 - x_0)$$

Summed-Area Tables

- Less blurry than MIP mapping
- How much storage does a summed-area table require?
  - Same as texture size
- Does it require more or less work per pixel than a MIP map?
  - Four texture lookups plus math
- Note that only MIP map is implemented by hardware and supported by OpenGL

No Filtering



MIP mapping

Summed-Area Table

Texture Synthesis

- Goal of Texture Synthesis: create new samples of a given texture
- Many applications: virtual environments, hole-filling, texturing surfaces

Texture Revisit

- Texture depicts spatially repeating patterns
- Many natural phenomena are textures

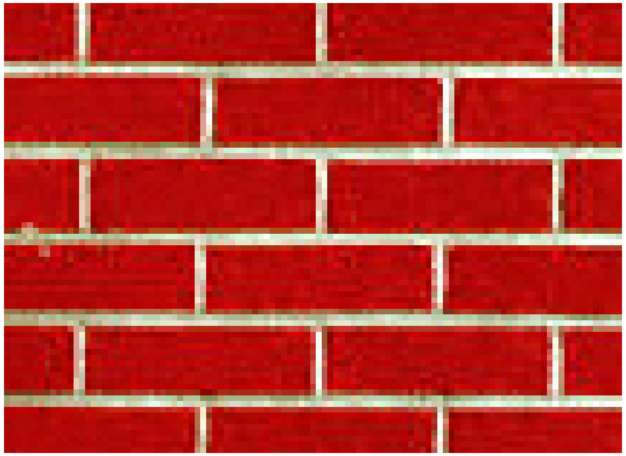
radishes

rocks

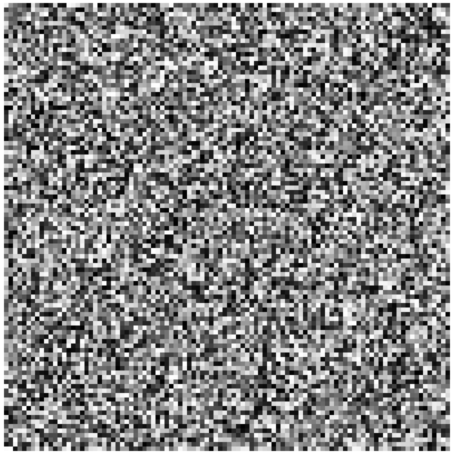
yogurt

Challenge in Texture Synthesis

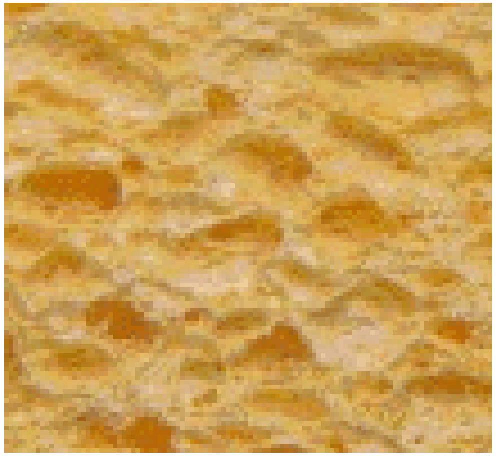
- Need to model the whole spectrum: from repeated to stochastic texture



Repeated

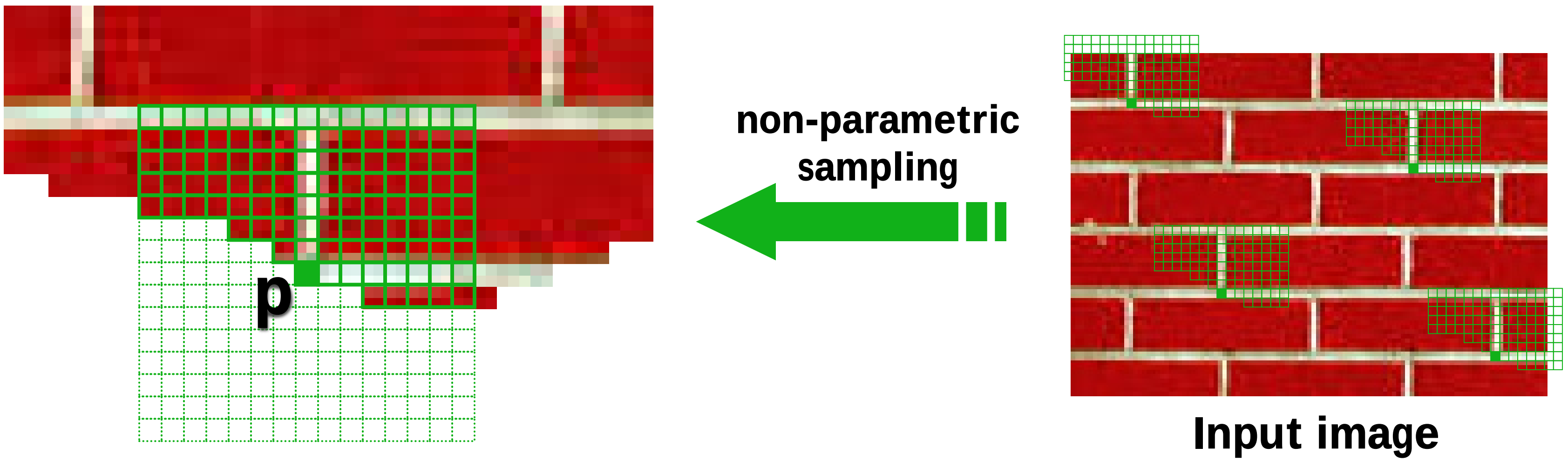


Stochastic



Both?

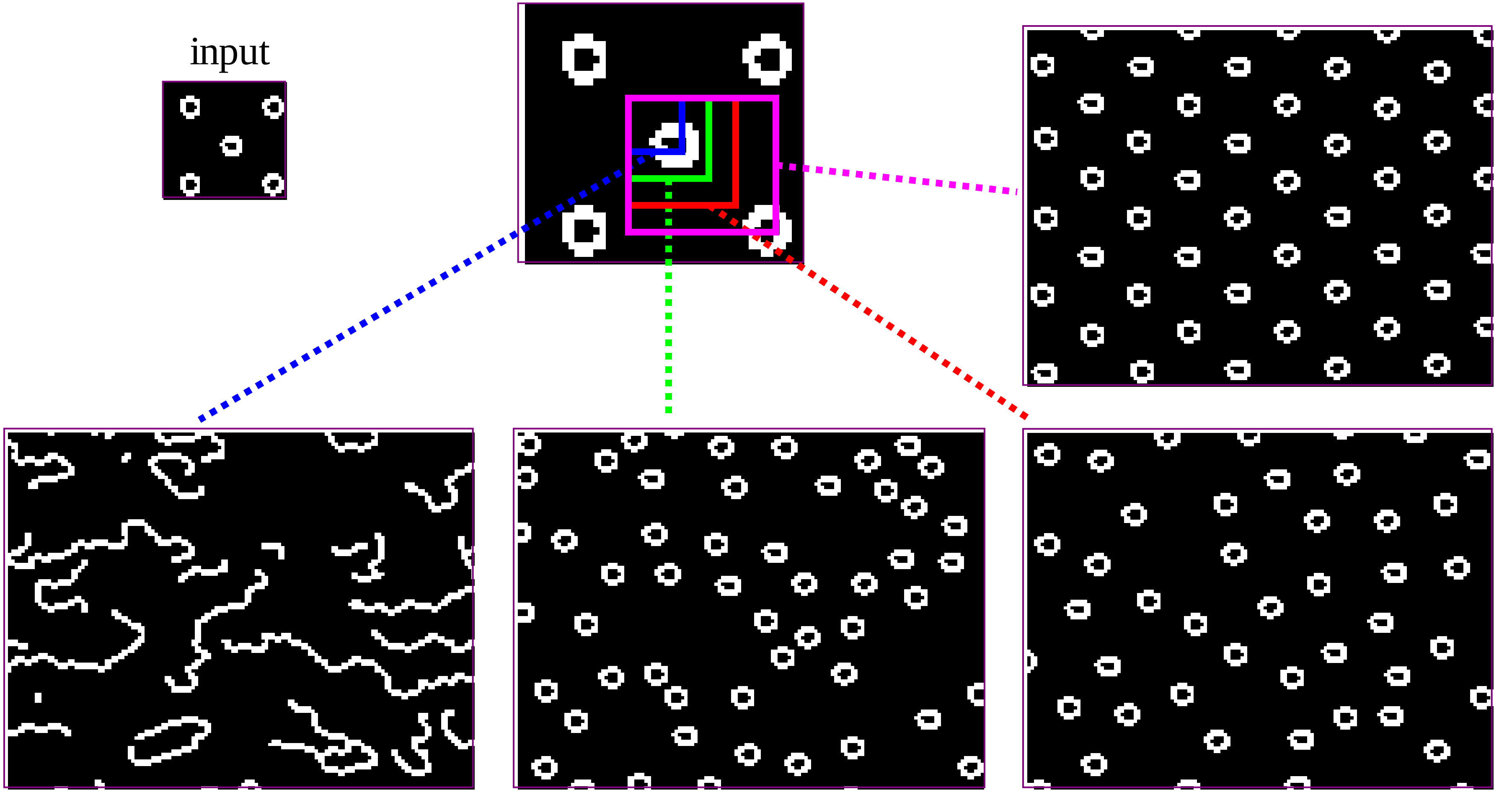
Efros & Leung Algorithm



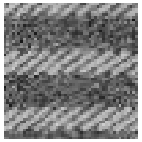
- Idea: Assuming Markov property, compute P(**p**|N(**p**))
  - Building explicit probability tables infeasible
  - Instead, we search the input image for all similar neighborhoods — that's our probability density function (pdf) for **p**
  - To sample from this pdf, just pick one match at random

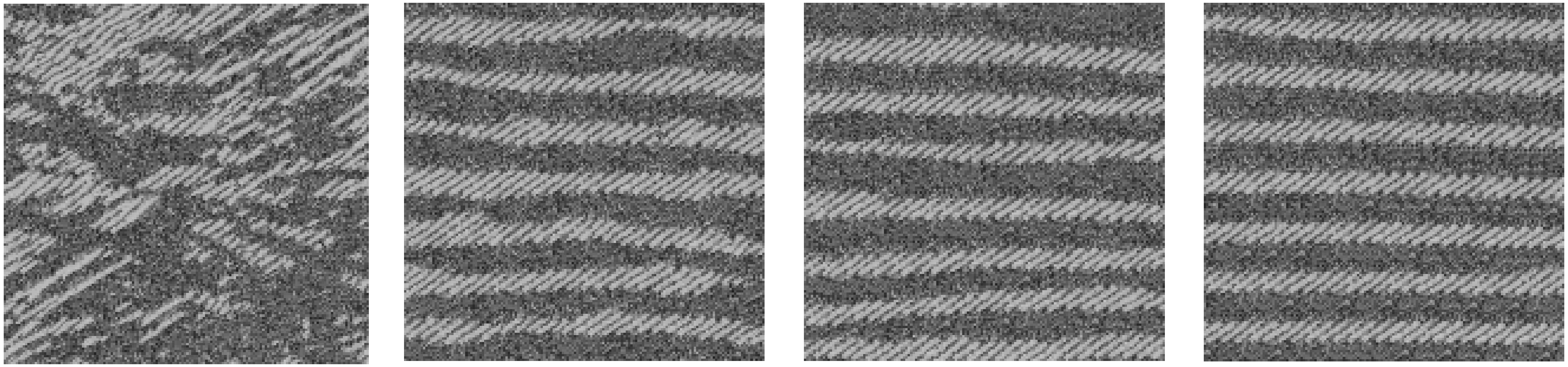
Reference: http://graphics.cs.cmu.edu/people/efros/research/EfrosLeung.html

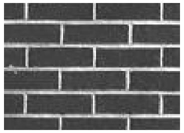
Neighborhood Window

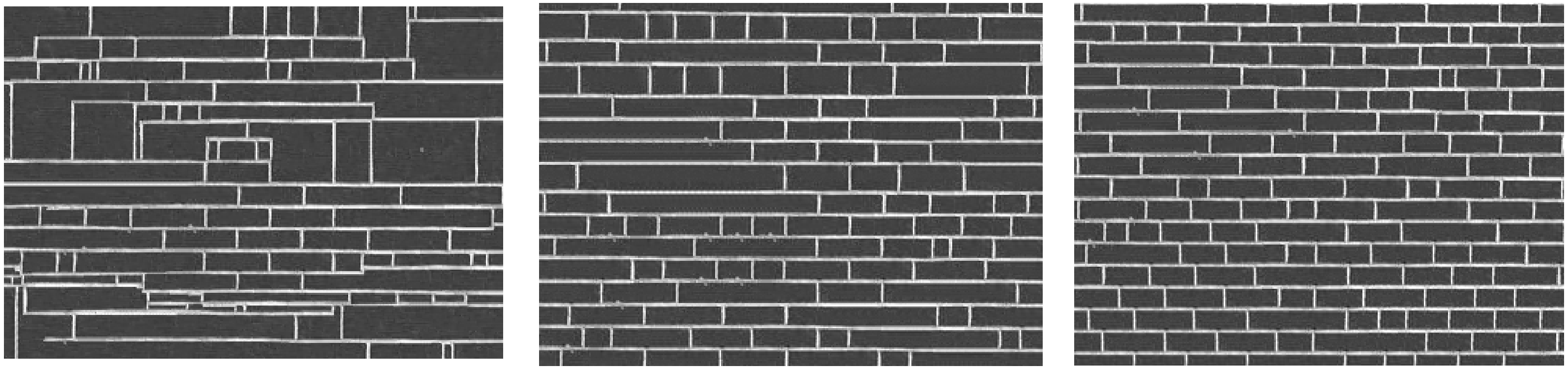


Varying Window Size





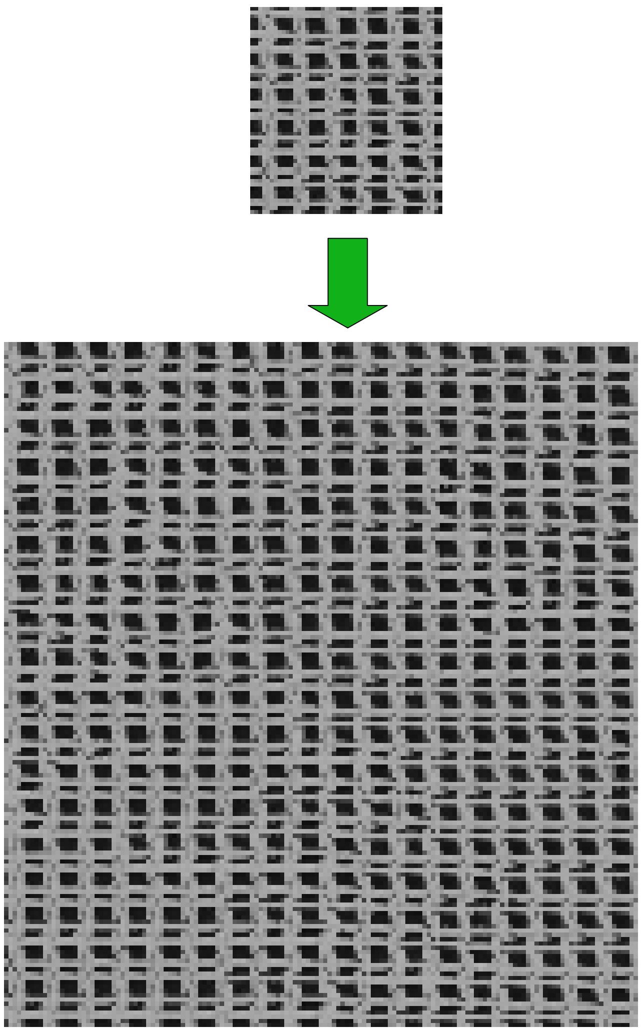




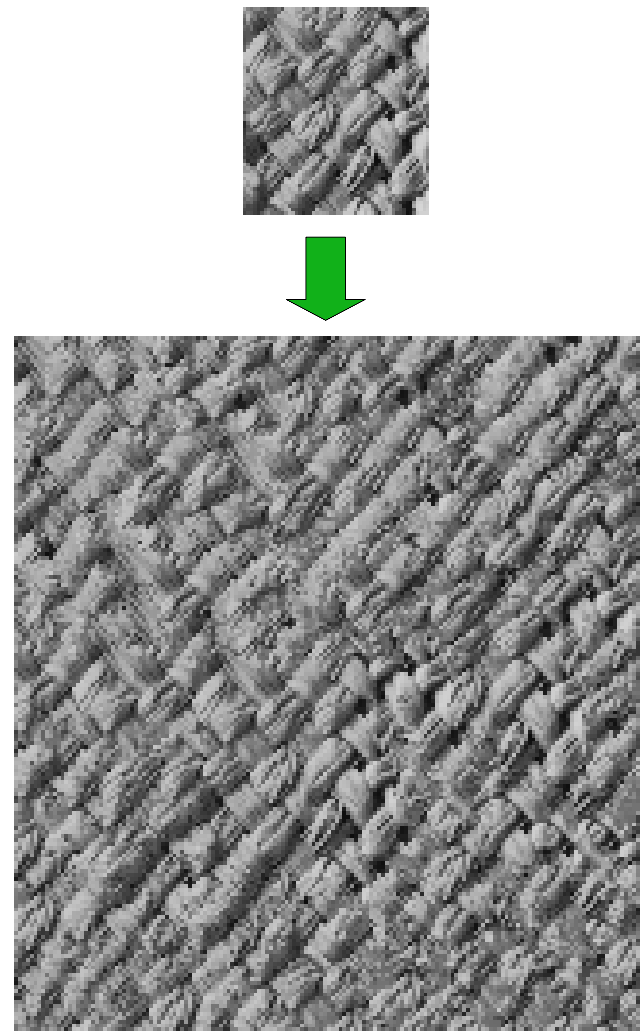
Increasing window size

Synthesis Results

french canvas

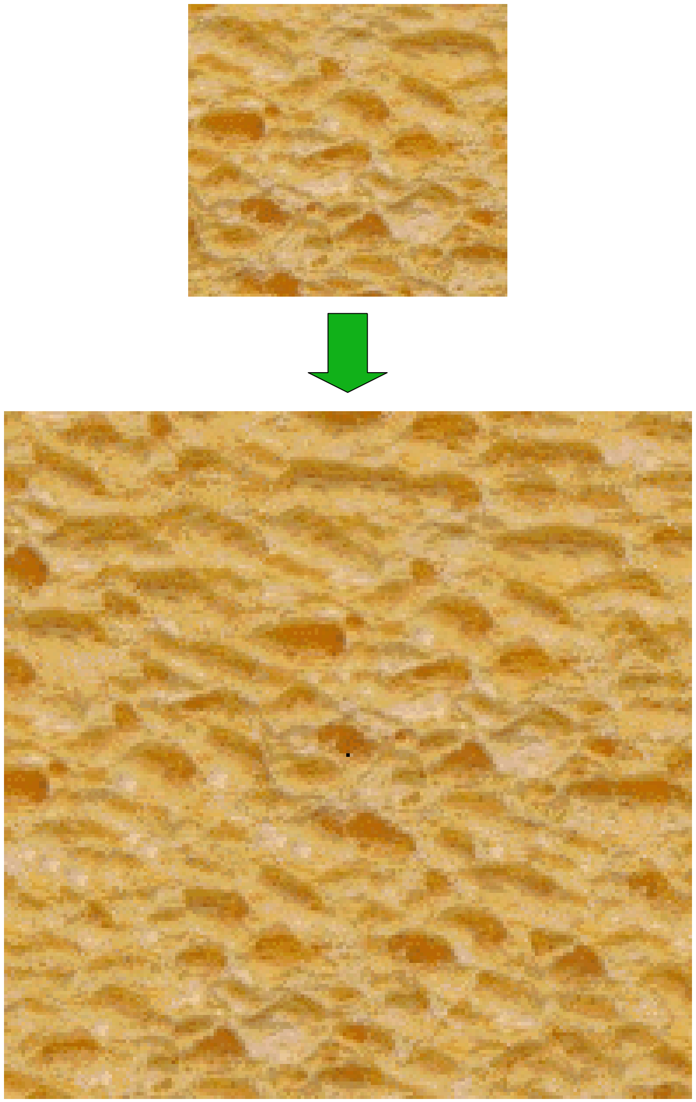


rafia weave

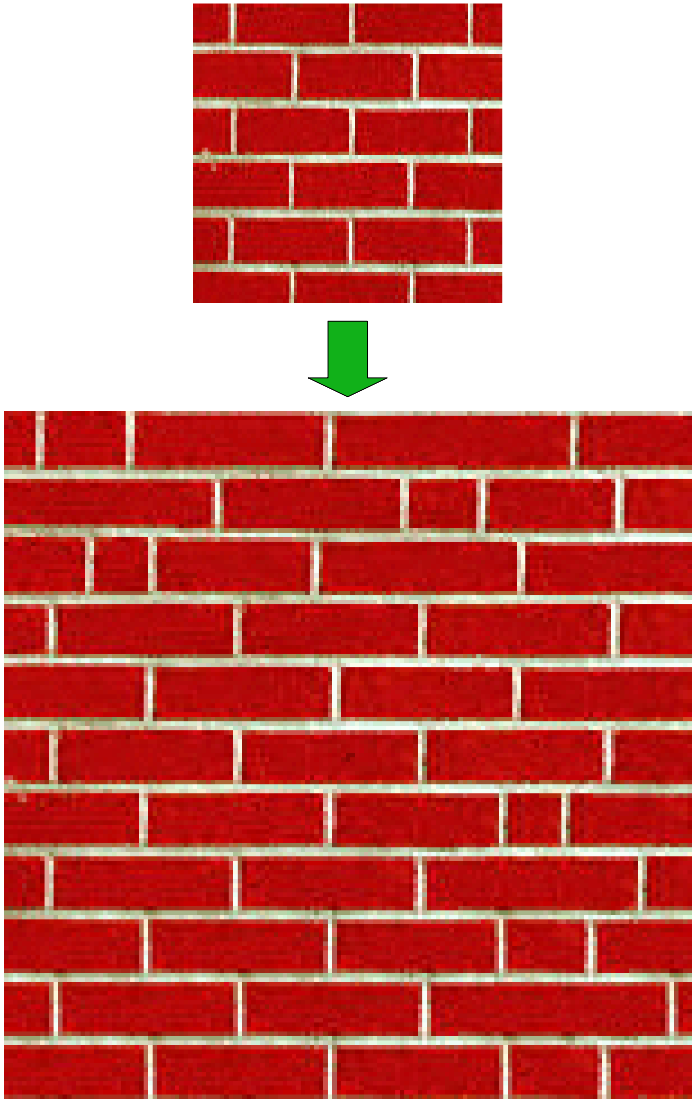


More Results

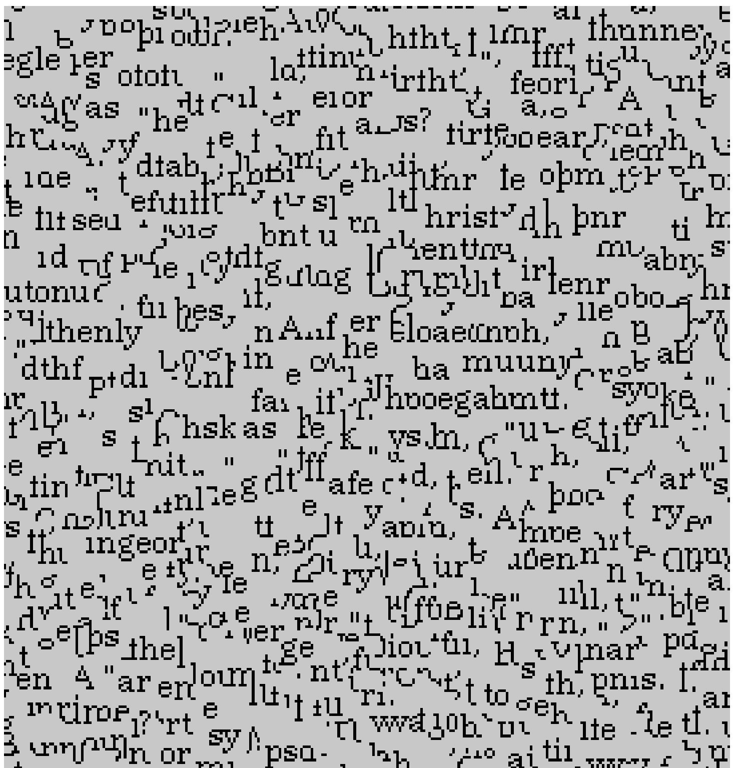
white bread

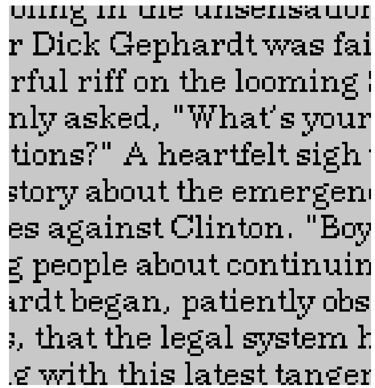


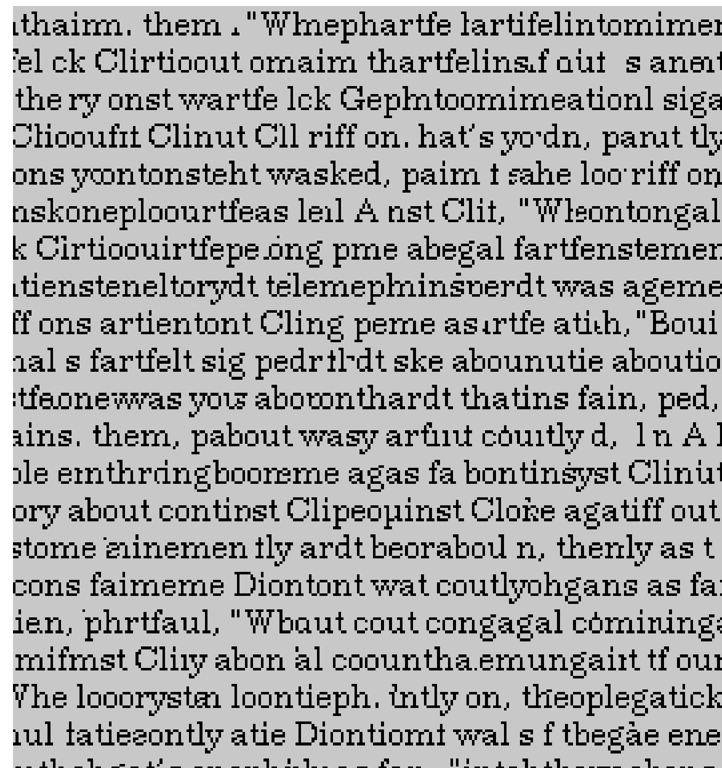
brick wall



Text Synthesis







Next Time

- Advanced texture mapping techniques
  - Shadow map
  - Bump map
  - Environment map
  - …