CPSC 314
Computer Graphics

Dinesh K. Pai

Texture Filtering (Ch. 16, 17, 18)

Some slides courtesy of M. Kim, KAIST

NOTICE:
Recordings of the lecture are provided to students enrolled in the course for self-study only. Any other use, including reproduction and sharing of links to materials, is strictly prohibited.

Preliminaries

- Announcements and Reminders
  - Monday March 14th class will also Quiz 2 Review. **Will be on Zoom!**
  - Wednesday guest lecture (on Zoom)
  - Friday March 18th Quiz 2, in-person in ANGU 098
- Today
  - Quiz 2 preparation notes
  - Mip-Mapping Wrap up
  - Aliasing revisited
  - Coverage and compositing (a first look)

Quiz 2 logistics

- Same rules as Midterm, Same location. Only change: No Type D question
  - On Canvas. Closed book, closed Internet except to access Quiz 2 page
  - Location: ANGU 098
  - Exam is for 50 minutes, during class
  - Budget 45 minutes for doing the quiz (One minute per mark)
  - 3 Types of Question (Parts A,B,C) as in Quiz 1
    - T/F questions
    - “Recognition” Fill in the blanks (with multiple choice).
    - “Computing” Solve a small problem, and select the correct answer.

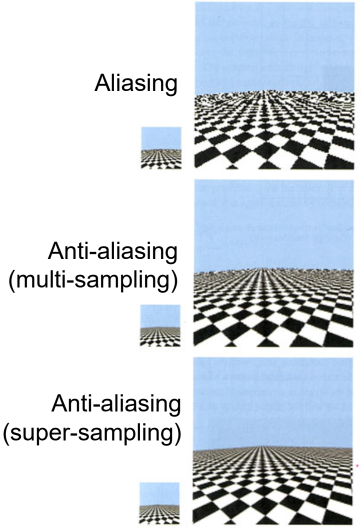
Quiz 2 Topics

- Everything covered in class through March 14th, and Assignment 3
- Textbook. Read ALL of these, except as noted
  - Ch 14.2-14.3 Rendering (parts covered after midterm, Assignment 3)
  - Ch 15 Texture Mapping (main focus of quiz)
  - Ch 16-18 Sampling, MipMapping (at the level we covered in class; no integrals required)
    also review
  - Ch 11 Depth Shadows Maps
  - Ch 12 Vertex to Pixel (focus on what is covered in class)
- NOTE: several topics are only covered in lectures (e.g., file formats, physically based rendering). Please follow lectures

  Topics from Quiz 1 and midterm will be assumed as pre-requisites (e.g., it is assumed you now know coordinate frames, transformations, reflection, etc.)

Recap: MipMapping

Problem: Aliasing



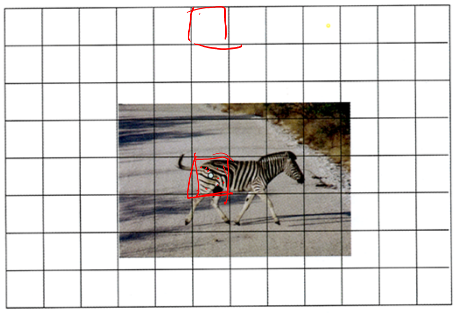
Aliasing

Anti-aliasing (multi-sampling)

Anti-aliasing (super-sampling)

Aliasing

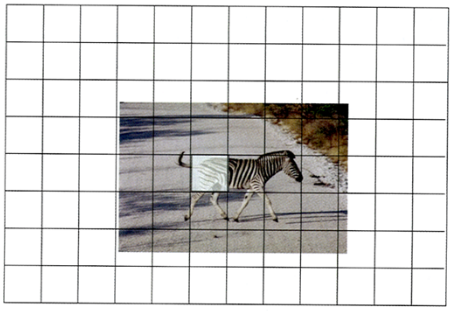
- The heart of the problem: too much information in one pixel



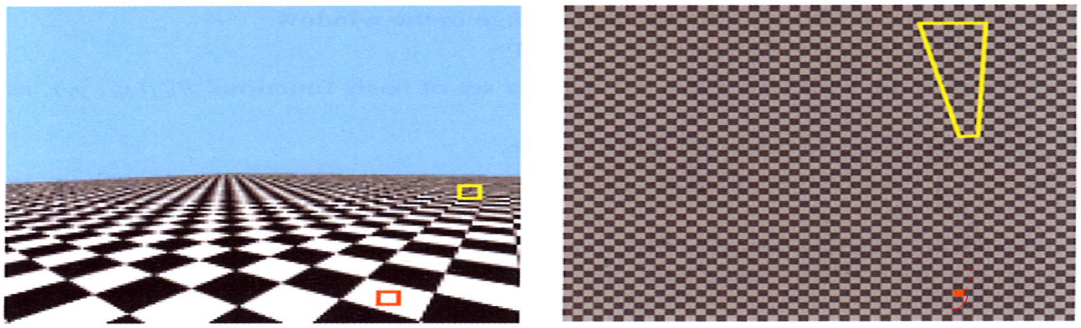
Soln: average or blur.

Anti-aliasing

- Intuitively: the single sample is a bad value, we would be better off setting the pixel value using some kind of average value over some appropriate region.
- In the above examples, perhaps some gray value.

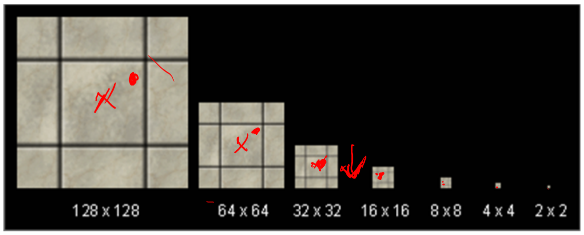


Resampling

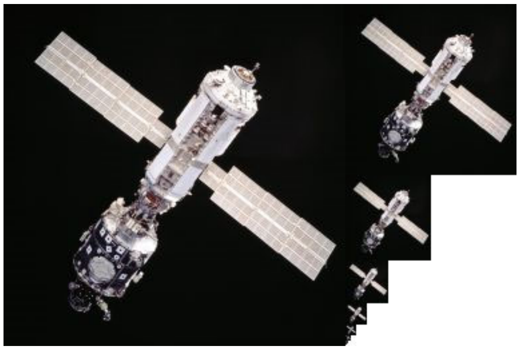


Mip mapping

- In mip mapping, one starts with an original texture T^0 and then creates a series of lower and lower resolution (blurrier) texture T^i.
- Each successive texture is twice as blurry. And because they have successively less detail, they can be represented with ½ the number of pixels in both the horizontal and vertical directions.



Mipmap example



Source: wikipedia

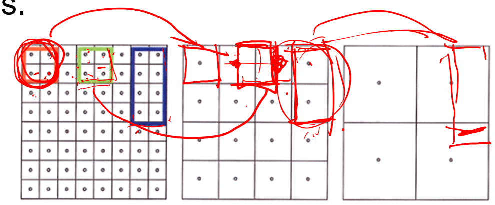
Useful Resource

- https://threejs.org/manual/#en/textures#filtering-and-mips ←

- Note: https://threejs.org/manual/ provides many other useful tutorials (used to be https://threejsfundamentals.org/). Check it out

Mip mapping

- In OpenGL/WebGL Mip mapping with trilinear interpolation is specified with the call glTexParameteri(GL TEXTURE 2D, GL TEXTURE MIN FILTER,GL LINEAR MIPMAP LINEAR)
- In Three.js set Texture.minFilter to THREE.LinearMipMapLinearFilter
- Trilinear interpolation requires WebGL to fetch 8 texture pixels and blend them appropriately for every requested texture access.

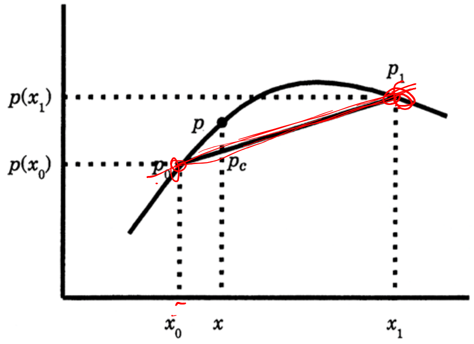


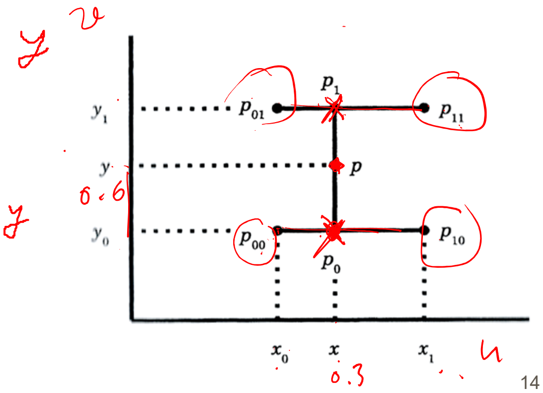
Recap: Linear and Bilinear interpolation

We already know how to interpolate in 1D

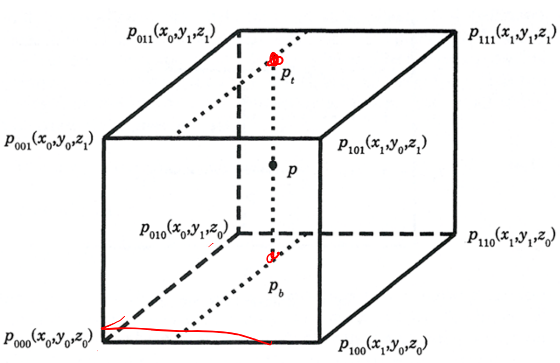
- Linear (1D)

Bilinear (2D):





Trilinear interpolation

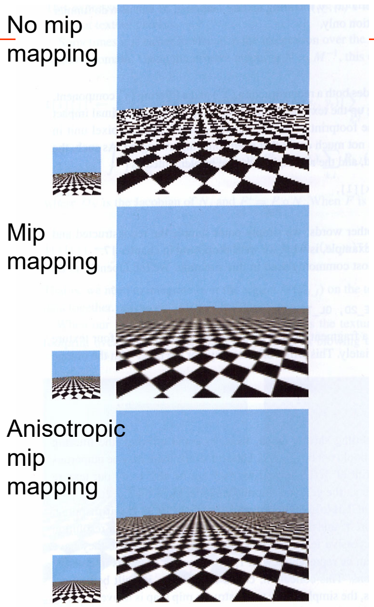


64x64

128x128

Properties

- It is easy to see that mip mapping works reasonably well, but has limitations that can be addressed by more advanced methods.

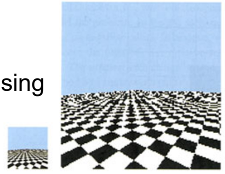


Magnification

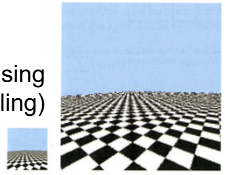
- We tell OpenGL to do this using the call glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR).
- In Three.js set Texture.magFilter to THREE.LinearFilter (default)
- For a single texture lookup in a fragment shader, the hardware needs to fetch 4 texture pixels and blend them appropriately.

Recap: Aliasing and anti-aliasing

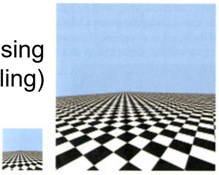
Aliasing



Anti-aliasing (multi-sampling)



Anti-aliasing (super-sampling)

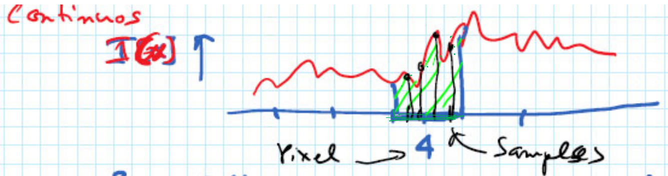


Aliasing

- Transforming sampled, image-like, data must be done carefully
  - Textures (input), Rasterization, Display (output)
- Naïve sampling can produce artifacts: Aliasing or “Jaggies”
- Need to “filter” the data to the appropriate resolution before discretization
  - Box filter is the easiest. **But how to compute this average efficiently?!**
  - Implemented using “over-sampling”
- Two types of over-sampling
  - Super-sampling
  - Multi-sampling

19

Sampling in 1D



Approximate with I[x]

Box filter: Average over a pixel

Oversampling: Approximate

$$\frac{\int I(x)}{\text{length}} \simeq \frac{\sum_{\substack{\text{samples}\\ \text{in pixel}}} I[x]}{N}$$

number of samples

20

Coverage

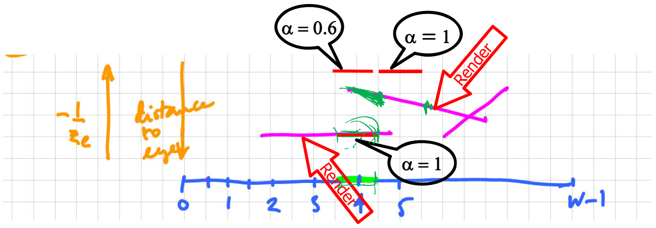
A first look

Coverage

- Rapid changes in color could be due to
  - Texture
  - Shading
  - Depth discontinuities
- Supersampling deals with all at once, but at great cost
- It may be more efficient to separately handle each of the sources of color change

Coverage

- Texture => Pre-filtered textures, “mip mapping”
- Shading => generally changes slowly, except at edges of triangles
- Depth discontinuities => check if discontinuity passes though pixel



Store (r g b α)

- Estimate partial coverage of pixel by triangle fragment
- Fraction of pixel covered is denoted alpha (α).

Alpha definition

- More specifically, let $I(x,y)$ be a continuous image, and let $C(x,y)$ be a binary valued (x,y) *coverage function*
  - C = 1 at any point where the image is “occupied”
  - C = 0 where it is not.
- Store in discrete image:

$$\mathrm{I[i][j]} \leftarrow \iint_{\Omega_{i,j}} I(x,y)C(x,y)dxdy$$

$$\alpha\mathrm{[i][j]} \leftarrow \iint_{\Omega_{i,j}} C(x,y)\,dx\,dy$$

NOTE: this is called “premultiplied alpha”

Multi-sampling

- *During the rasterization* of each triangle, “coverage” and z-values are computed at “high resolution”.
- But for efficiency, the fragment shader is only called **only once per final resolution pixel.**
  - This color data is shared between all of the samples hit by the triangle in a single (final resolution) pixel.
- Once rasterization is complete, groups of these high resolution samples are averaged together.

Compositing?

- Example of demo reel http://vimeo.com/72617082