Sampling

Dinesh K. Pai

Textbook Chapter 16

Several slides courtesy of M. Kim

1

Today

- Announcements
  - Reminder: Quiz 3 on Friday
  - I will post a couple of Quiz 3 practice questions on Piazza today. Will discuss answers on Wednesday
- Projector Texture mapping tips (need for A4)
- Sampling and Aliasing

2

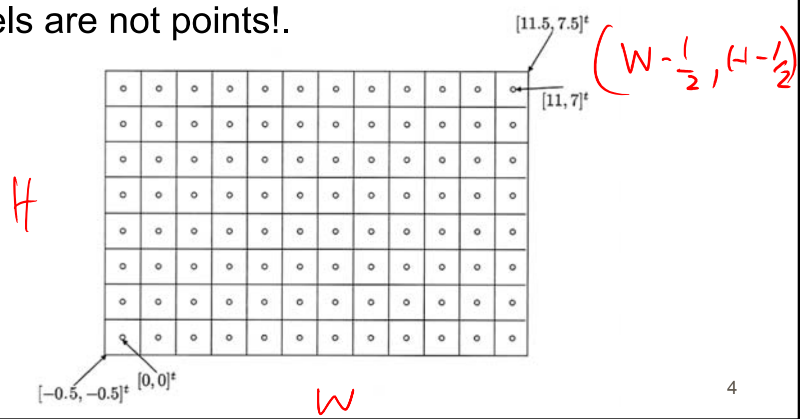
Projector texture mapping tips

- Read Texture Viewport (Textbook 12.3)
- Check out this excellent demo of transformations: http://www.realtimerendering.com/udacity/transforms.html

3

Viewport

- Convention in text: pixel centers are integers.
  - Warning: OpenGL docs usually assume bottom left corner of each pixel has integer coordinates.
- Pixels are not points!.



4

Viewport matrix

- We need a transform that maps the lower left corner to $[-0.5,-0.5]^t$ and upper right corner to $[W-0.5, H-0.5]^t$
- The appropriate scale and shift can be done using the viewport matrix:

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \begin{bmatrix} W/2 & 0 & 0 & (W-1)/2 \\ 0 & H/2 & 0 & (H-1)/2 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \\ z_n \\ 1 \end{bmatrix}$$

5

Sampling

6

Two views of images

- A *continuous image,* $I(x_w, y_w)$, is a bivariate function.
  - range is a linear color space.
- A *discrete image* I[i][j] is a two dimensional array of color values.
- We associate each pair of integers i, j, with the continuous image coordinates $x_w = i \text{ and } y_w = j$

7

Sampling

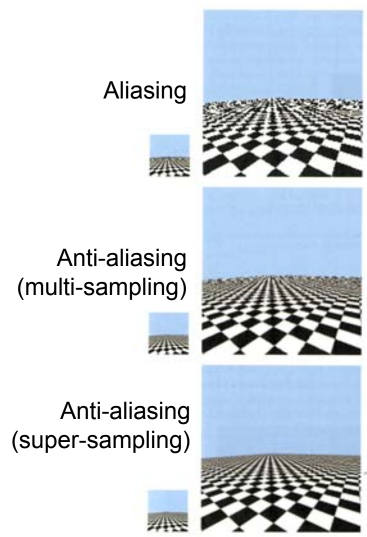
- The simplest and most obvious method to go from a continuous to a discrete image is by *point sampling.*
- To obtain the value of a pixel i, j, we sample the continuous image function at a single integer valued domain location:

$$\mathrm{I[i][j]} \leftarrow I(i, j)$$

- This can results in unwanted artifacts.

8

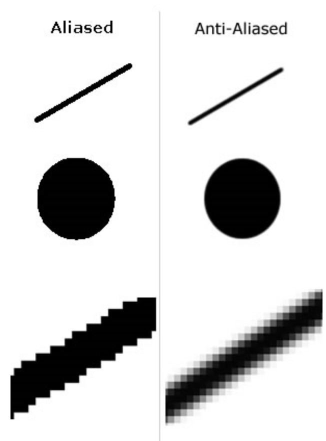
Aliasing and anti-aliasing



9

Aliasing

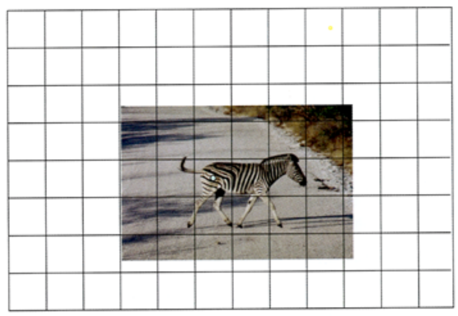
- Scene made up of black and white triangles: jaggies at boundaries
  - Jaggies will crawl during motion
- If triangles are small enough then we get random values or weird patterns.



10

Aliasing

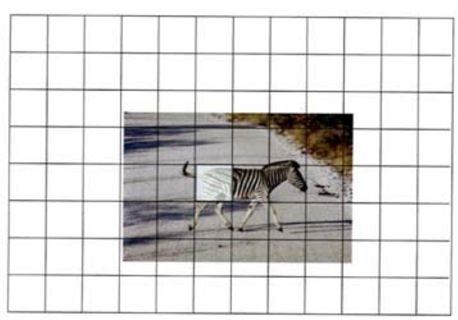
- The heart of the problem: too much information in one pixel



11

Anti-aliasing

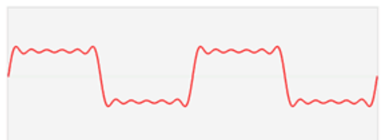
- Intuitively: the single sample is a bad value, we would be better off setting the pixel value using some kind of average value over some appropriate region.
- In the above examples, perhaps some gray value.



12

Anti-aliasing

- Mathematically this can be modeled using *Fourier analysis*.
  - Breaks up the data by “frequencies” and figures out what to do with the un-representable high frequencies.



13

Anti-aliasing

- We can also model this as an optimization problem.
- These approaches lead to:

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

- where $F_{i,j}(x,y)$ is some function that tells us how strongly the continuous image value at $[x,y]^t$ should influence the pixel value $\mathrm{i, j}$

14

Anti-aliasing

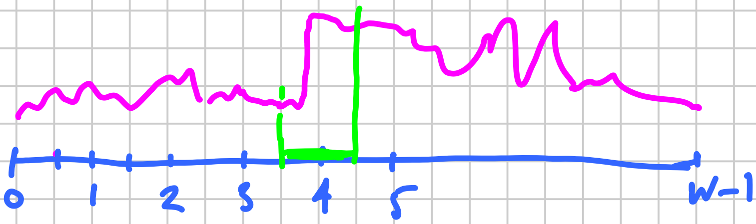
- In this setting, the function $F_{i,j}(x,y)$ is called *a filter*.
  - In other words, the best pixel value is determined by performing some continuous weighted averaging near the pixel's location.
  - Effectively, this is like blurring the continuous image before point sampling it.

15

- Switch to tablet

16

Images in 1D



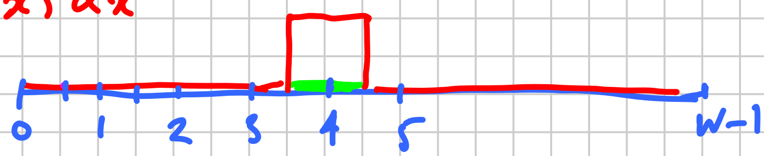
$I(x)$ continuous image.

$I[i]$ discrete image

extent of pixel

Filter $\approx$ weighted average

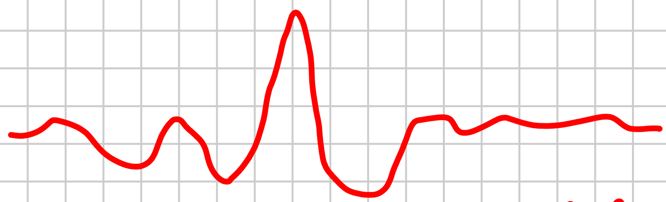
$\int I(x)\,F(x)\,dx$



$F(x)$ = Box Filter

$\equiv$ uniformly weighted average in a single pixel

= if inside pixel, average

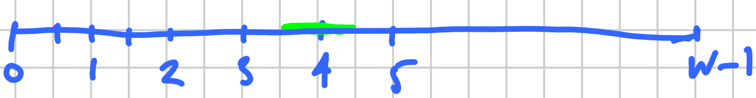


Optimal filters may look like this (e.g. sinc)

Oversampling

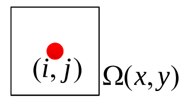
Approximate

$$\int I(x)\,F(x)\,dx = \sum I[s_i]\,/N$$



Box filter

- We often choose the filters $F_{i,j}(x,y)$ to be something non-optimal, but that can more easily computed with.
- The simplest such choice is a *box filter*, where $F_{i,j}(x,y)$ is zero everywhere except over the 1-by-1 square center at $x = i, y = j$.



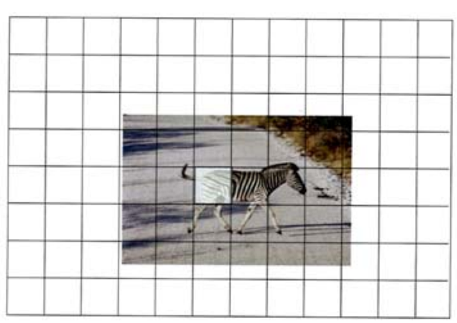
- Calling this square $\Omega_{i,j}$, we arrive at

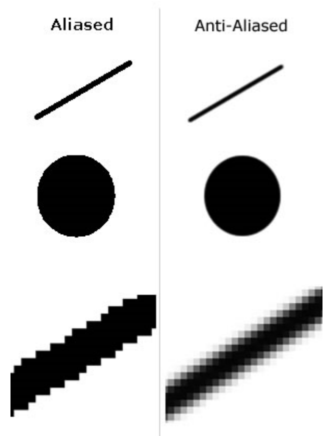
$$\mathrm{I[i][j]} \leftarrow \iint_{\Omega_{i,j}} I(x,y)dxdy$$

17

Box filter

- In this case, the desired pixel value is simply the average of the continuous image over the pixel's square domain.



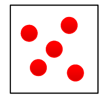


18

Over-sampling

- Even that integral is not really easy to compute
- Instead, it is approximated by some sum of the form:

$$\mathrm{I[i][j]} \leftarrow \frac{1}{n}\sum_{k=1}^{n} I(x_k, y_k)$$

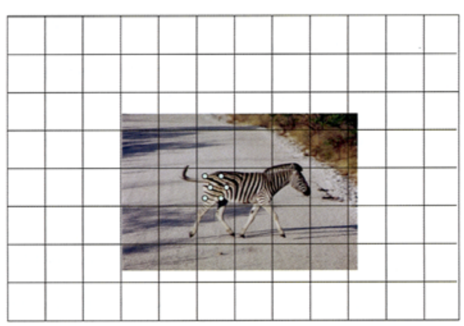


  where *k* indexes some set of locations (x_k, y_k) called the sample locations.
- The renderer first produces a “high resolution” color and z-buffer “image”,
  - where we will use the term *sample* to refer to each of these high resolution pixels.

19

Over-sampling

- Then, once rasterization is complete, groups of these samples are averaged together, to create the final lower resolution image.



20

Super-sampling

- If the sample locations for the high resolution image form a regular, high resolution grid, then this is called *super sampling.*
- We can also choose other sampling patterns for the high resolution "image",
  - Such less regular patterns can help us avoid systematic errors that can arise when using the sum to replace the integral.

21

Multi-sampling

- Render to a "high resolution" color and z-buffer
- *During the rasterization* of each triangle, "coverage" and z-values are computed at this sample level.
- 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.

22