Antialiasing

UCSD CSE 167
Tzu-Mao Li
Recall: aliasing

// back to front
for each primitive
for each pixel in bounding box
if pixel center hits primitive:
  color = primitive.color

"aliasing"
Aliased vs antialiased
Recall: Super-sampling Antialiasing (SSAA)

combat aliasing artifacts by taking average of color inside a pixel
Pseudocode of SSAA

// n = number of samples per pixel
img = Image(n * w, n * h)
Z_buffer = Image(n * w, n * h)
for each triangle t:
  for each pixel x, y:
    for each sample x', y':
      if x', y' not in t: continue
      z = … // depth interpolation
      c = fragment_shader(t at x', y')
      if z < Z_buffer[x', y']:
        Z_buffer[x', y'] = z
        img[x', y'] = c
img = downsample(img, n)
Sidetrack: Early Fragment Test

modern GPUs, when possible, will skip fragments if its Z value fails depth test

```
for each triangle t:
    for each pixel x, y:
        for each sample x', y':
            if x', y' not in t: continue
            z = ... // depth interpolation
            c = fragment_shader(t at x', y')
            if z < Z_buffer[x', y']:
                Z_buffer[x', y'] = z
                img[x', y'] = c
...```

```
for each triangle t:
    for each pixel x, y:
        for each sample x', y':
            if x', y' not in t: continue
            z = ... // depth interpolation
            if z >= Z_buffer[x', y']:
                continue
            if z < Z_buffer[x', y']:
                Z_buffer[x', y'] = z
            c = fragment_shader(t at x', y')
            img[x', y'] = c
...```

see [https://www.khronos.org/opengl/wiki/Early_Fragment_Test](https://www.khronos.org/opengl/wiki/Early_Fragment_Test) for detail
Pseudocode of SSAA

// n = number of samples per pixel
img = Image(n * w, n * h)
z_buffer = Image(n * w, n * h)
for each triangle t:
  for each pixel x, y:
    for each sample x', y':
      if x', y' not in t: continue
      z = ... // depth interpolation
      if z >= z_buffer[x', y']: continue
      z_buffer[x', y'] = z
      c = fragment_shader(t at x', y')
      img[x', y'] = c
  img = downsample(img, n)
Speeding up antialiasing by approximation

observation: the fragments in the pixel that hit the same triangle have similar colors!

```
// n = number of samples per pixel
img = Image(n * w, n * h)
z_buffer = Image(n * w, n * h)
for each triangle t:
    for each pixel x, y:
        for each sample x', y':
            if x', y' not in t: continue
            z = ... // depth interpolation
            if z >= z_buffer[x', y']: continue
            z_buffer[x', y'] = z
            c = fragment_shader(t at x', y')
            img[x', y'] = c
img = downsample(img, n)
```
Multi-sample antialiasing (MSAA) [Akeley 1993]

do Z test on multiple samples to compute “coverage”,
execute fragment shader only once per triangle

// n = number of samples per pixel
img = Image(n * w, n * h)
Z_buffer = Image(n * w, n * h)
for each triangle t:
    for each pixel x, y:
        bitmask = 0
        for each sample x', y':
            if x', y' not in t: continue
            z = ... // depth interpolation
            if z >= Z_buffer[x', y']: continue
            Z_buffer[x', y'] = z
            bitmask[x', y'] = 1
        if no visible sample: continue
        c = fragment_shader(t at x, y)
        for each sample x', y':
            if bitmask[x', y']:
                img[x', y'] = c
    img = downsample(img, n)
MSAA sampling patterns

https://www.overclockersclub.com/reviews/serious_statistics_aliasing/5.htm
MSAA sampling patterns

SSAA vs MSAA

https://eyesofthebeast.com/2015/01/wows-patch-6-1-anti-aliasing-options/
SSAA vs MSAA

here, the leaf is rendered using “alpha testing” using an alpha map texture inside fragment shader

https://eyesofthebeast.com/2015/01/wows-patch-6-1-anti-aliasing-options/
SSAA vs MSAA

MSAA does poorly when there are specular highlights

https://vr.arvilab.com/blog/anti-aliasing
Coverage-sampling antialiasing (CSAA) [NVIDIA] and Enhanced-quality antialiasing (EQAA) [ATI] add “coverage samples” that store bit coverage and not color nor depth.

- **depth sample**: test both depth and coverage, store color & depth of the closest triangle.
- **coverage sample**: test coverage (whether a point is inside the triangle).
- **color sample**: for each triangle, compute the fragment color here.

Not fully clear how exactly it’s implemented.

Morphological Antialiasing (MLAA) [Reshetov 2009]:
antialiasing as image processing

much faster than MSAA, tend to produce blurrier images

render without antialiasing
postprocess to make it smooth
MLAA idea: edge detection -> smooth
MLAA idea: edge detection -> smooth
Many variants of MLAA

mostly just differ in how they detect edges and blur images

Fast approXimate Antialiasing (FXAA) [Lottes 2009]
Subpixel Morphological Antialiasing (SMAA) [Jimenez et al. 2012]
Conservative Morphological Antialiasing (CMAA) [Stugar]

all of them are widely used in many video games
MSAA vs postprocessing AA

edge detection is not always reliable
MSAA vs postprocessing AA

blurring too much

not blurring enough
Postprocessing AA can be combined with SSAA/MSAA
Temporal antialiasing (TAA)
[Yang et al. 2009]

idea: accumulate samples from previous frames

Frame N-1

- Green pixels: in previous output frame
- Yellow dots: Accumulated temporal samples (not individually stored)

Frame N

- Orange circle: Target pixel
- Blue dot: New shading sample

http://behindthepixels.io/assets/files/TemporalAA.pdf
for each pixel sample, compute how much the object has moved
(usually can get this from the animation pipeline)
TAA steps

find the corresponding pixel locations in the previous frame

frame n-1

frame n
TAA steps

\[ P_n = \alpha C_n + (1 - \alpha)P_{n-1} \]

- **blend factor**
- **previous frame resolved color**
- **current frame**
- **new color sample**
- **resolved color**
- **blend with previous frame color**
TAA in Unreal Engine 4

No Temporal AA
What if the motion detection is wrong?

e.g., occlusion between triangles
Many tricks to reject/rectify the temporal samples

• reject old samples by testing depth, normals, object/material IDs, etc

• “neighborhood clipping”: only blend with similar colors

• very hard to reliably detect all changes

When fail, it leads to “ghosting”
TAA ghosting artifacts

Is there a way to fix this issue with TAA? It occurs whenever something on screen moves fast. I just moved my camera around really fast and this ghosting effect shows up. It doesn't appear without anti-aliasing or with FXAA. I know this happens in other games too, but is there a way to fix it?
TAA ghosting artifacts
Deep Learning Supersampling (DLSS) or Deep Learning Antialiasing (DLAA): TAA with neural networks
Deep Learning Supersampling (DLSS) or Deep Learning Antialiasing (DLAA):
TAA with neural networks

very expensive
only worth it when the scene is very difficult to render

neural networks

antialiased output
Deep Learning Supersampling (DLSS) or Deep Learning Antialiasing (DLAA):
TAA with neural networks
Next: non-photorealistic rendering