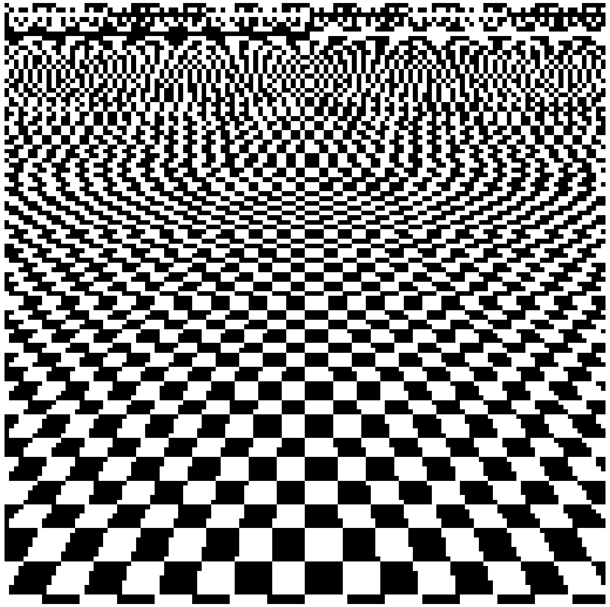
GRAPHICS GROUP

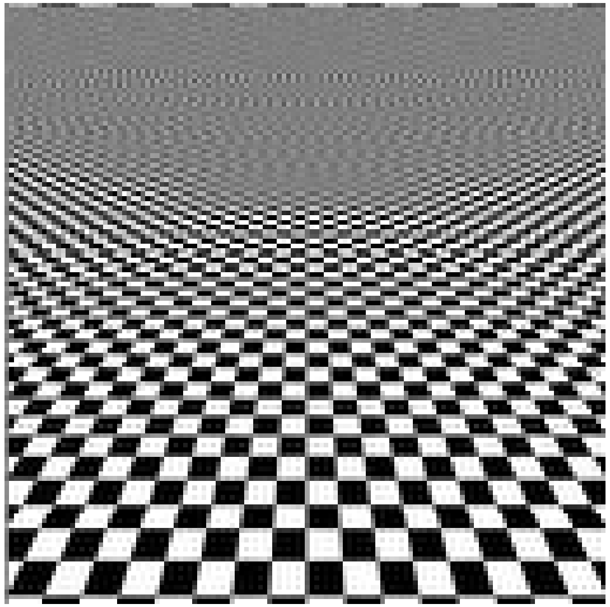
Computer Graphics: Rasterization II

Part 2 – Lecture 13

Today’s Outline

- Anti-Aliasing
  - Prefiltering
  - Postfiltering
  - Supersampling
- Drawing Text in OpenGL



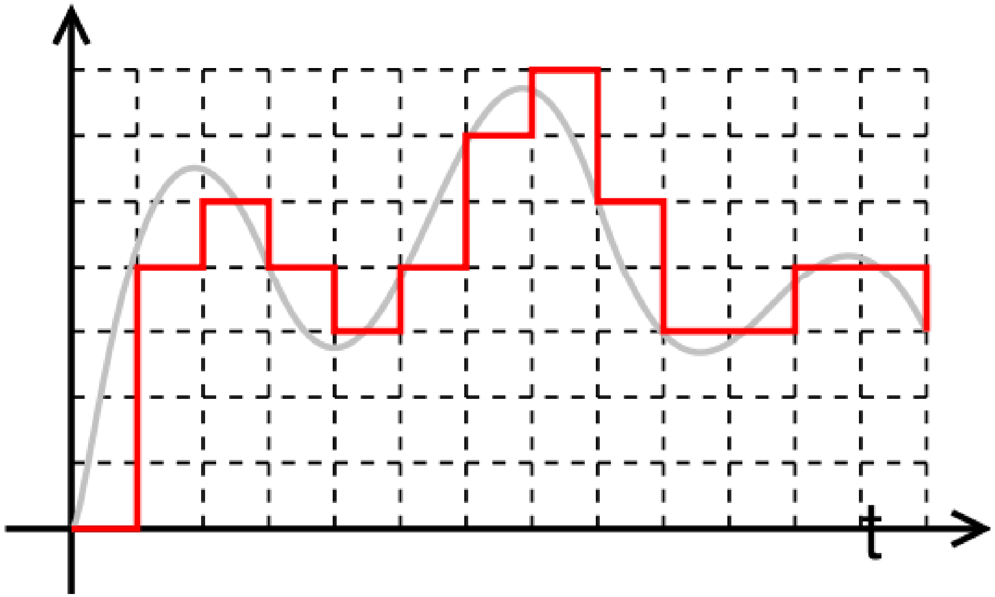


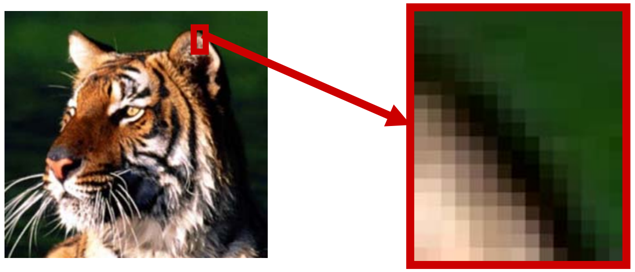
ANTI-ALIASING

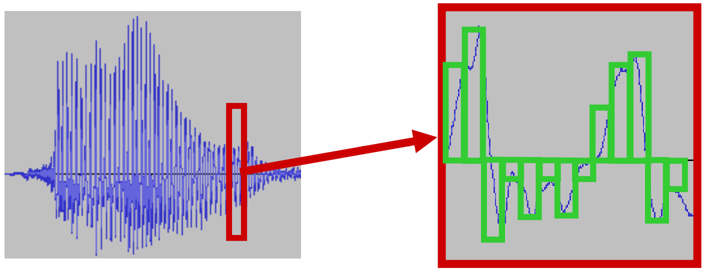
Images thanks to Sébastien Loisel

Sampling and Quantization

- **Sampling**: reducing a continuous (or very fine-grained) signal to a discrete (or more coarse-grained) signal by taking samples of it
- **Quantization**: approximating a continuous range (or very large set) of values with a smaller set of discrete values
- Both needed to represent real-world information digitally
- However: means loss of information



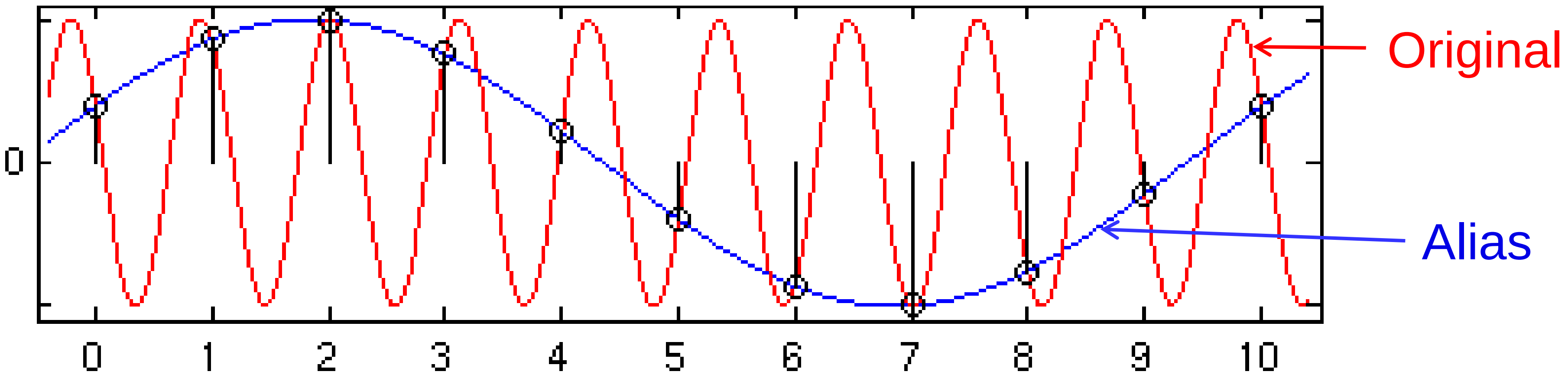




Aliasing

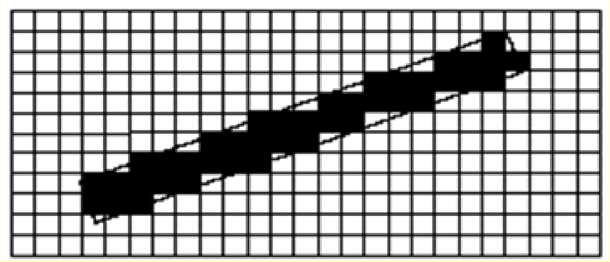
A signal looks like another signal (the "alias") after sampling

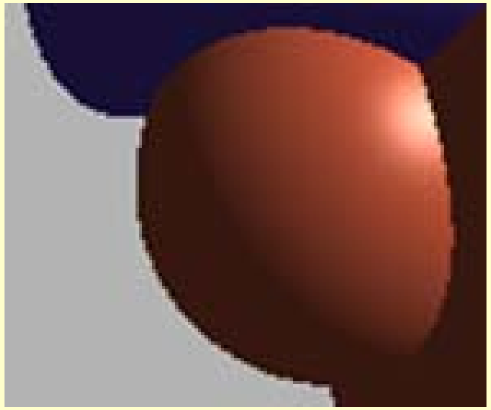
- Not a problem if the signals are still very similar
- But is a problem if the alias looks really different (→ aliasing artifacts)
- Happens particularly when sampling a high-frequency signal with a low sample frequency

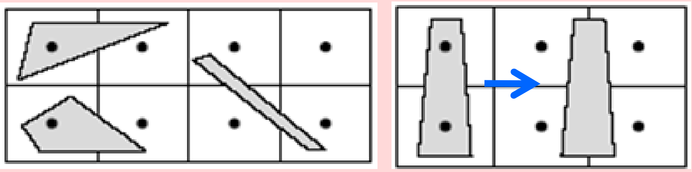


Aliasing Examples

“Jaggies” on edges

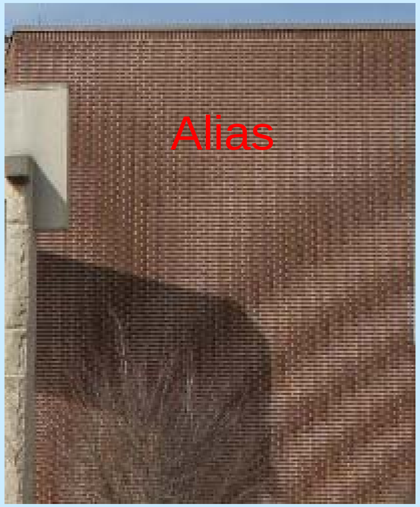






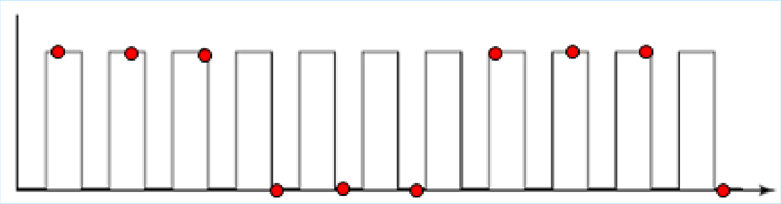
Small objects missed

Moiré patterns when sampling repetitive signals (e.g. textures)



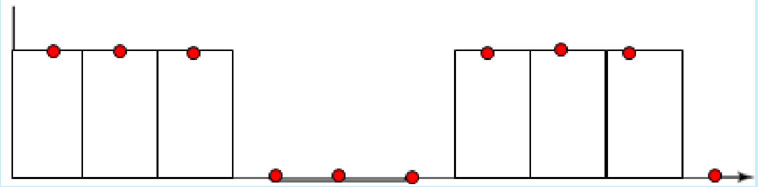
High-freq. signal

■□■□■□■□■□



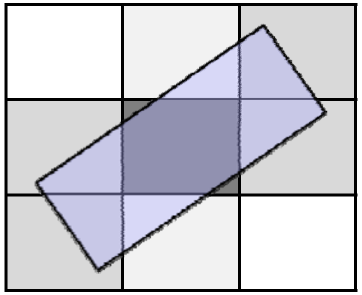
Low-freq. artifact

□□□■



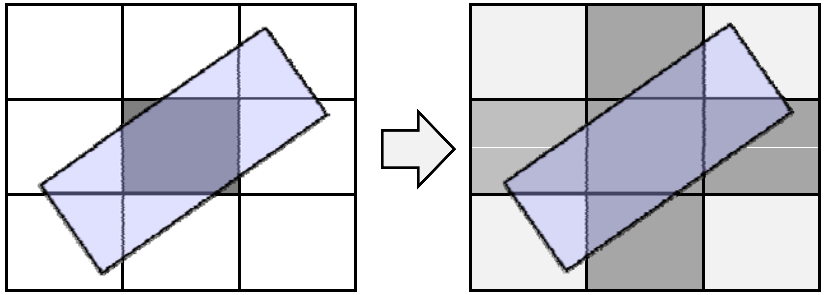
Anti-Aliasing

Trying to avoid that the sampled signal looks too much like a completely different signal (an “alias”)

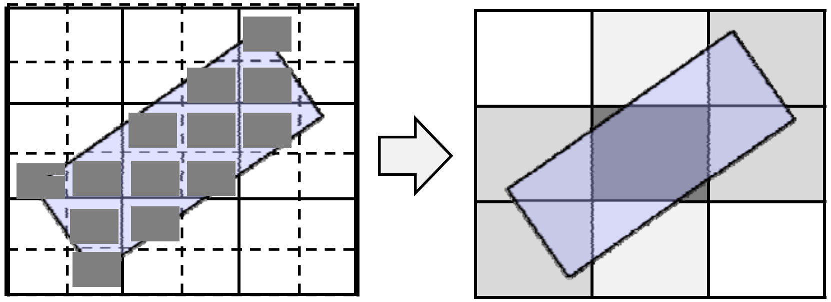


1. **Prefiltering**: determine actual coverage of objects visible in a pixel, and weigh object color by coverage

2. **Postfiltering**: smooth image by calculating pixels as weighted sum of several pixels



3. **Supersampling**: increase the number of samples per pixel, perform postfiltering over subpixels



Sampling Filters

Weighting function for averaging around a sample point

- Applied by performing a convolution operation:
  1. Place kernel center on the pixel to filter
  2. Multiply pixel values with corresponding kernel values
  3. Sum up and normalize (sum of weights should be 1)
- Reduces artifacts (esp. jaggies) but also blurs the image

Box filter

- Average in a square region around each pixel
- Kernel is filled with same value everywhere
- Rather poor quality, but ok for reducing jaggies

1/9 ·

1	1	1
1	1	1
1	1	1

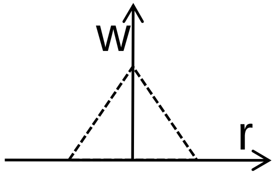
Kernel

High-Quality Filters

1. Weights of high-quality filters drop off radially
2. Better to average over a larger neighborhood

Bartlett filter

- Pixels closer to the center weigh more
- Like placing a cone onto the kernel (height = relative weight)

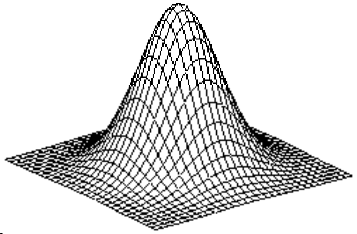


1/16 ·

1	2	1
2	4	2
1	2	1

Gaussian Filter

- Pixels near the center weigh more according to Gauss function
- Like placing 3-dimensional bell curve onto the kernel



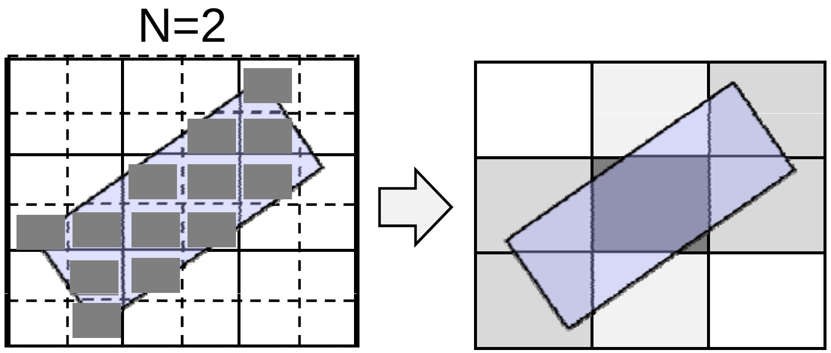
$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Supersampling

Sample more pixels than are actually visible (subpixels), then average over them (using a filter)

- Compute N samples in x and y for each screen pixel
- Approximates prefiltering



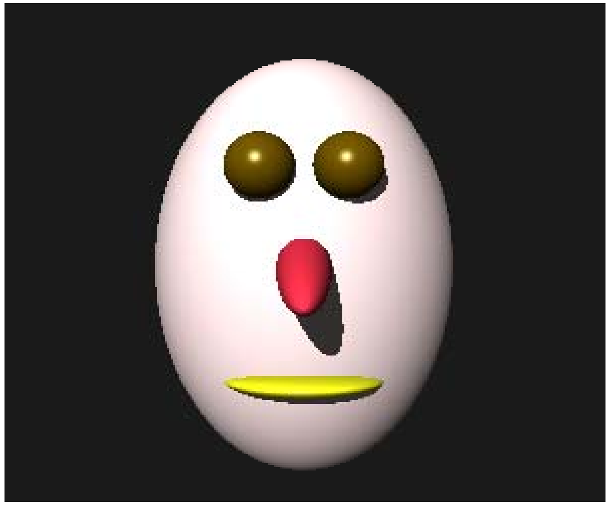
Advantages

- Less jaggies
- Can also capture small objects

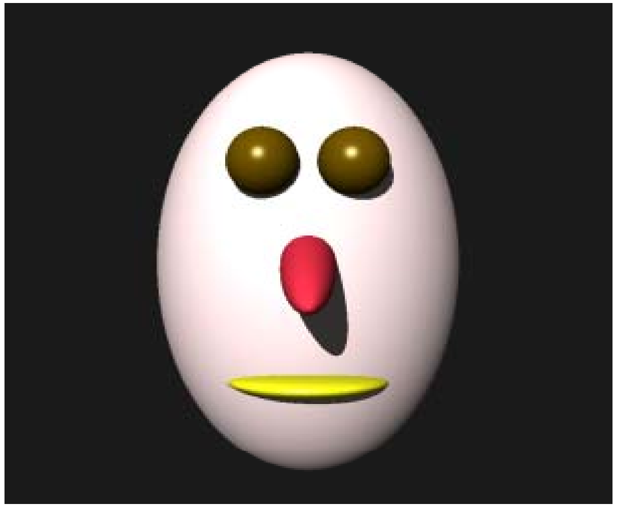
Disadvantages

- Expensive (N^2 times as many pixels to compute)
- Doesn't eliminate Moiré because samples are still uniformly spaced

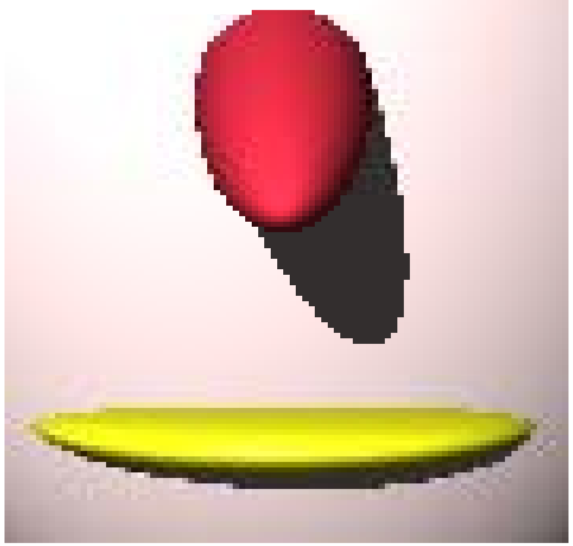
Anti-Aliasing Example

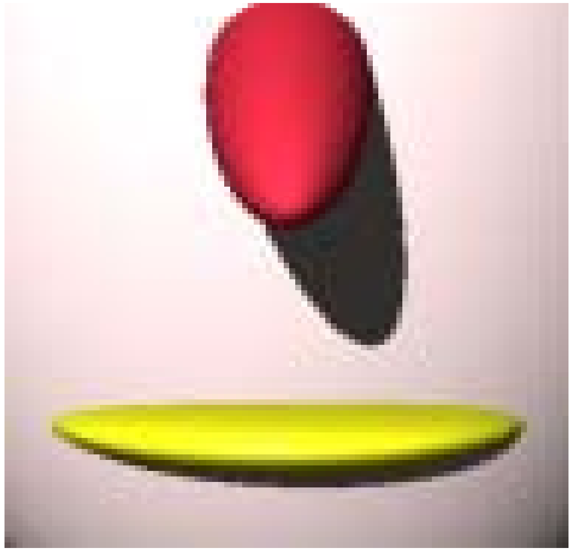


No anti-aliasing



Simple 3 x 3 supersampling

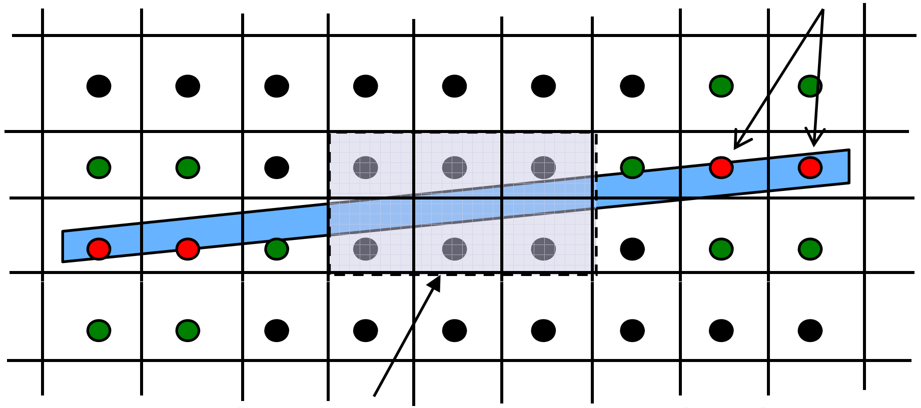




Adaptive Supersampling

Use supersampling only where it is needed

- Supersample only if high variance between adjacent pixels, e.g. if difference between pixel and its 4 neighbors exceeds a threshold
- Can be done recursively, i.e. supersample subpixels again
- Big performance gain (commonly used in ray tracing)
- But still Moiré patterns and other artefacts (e.g. small objects that disappear during animation)



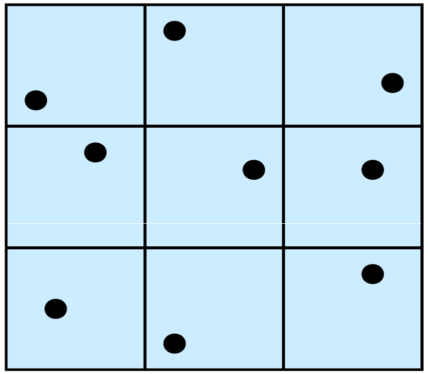
Supersampling here

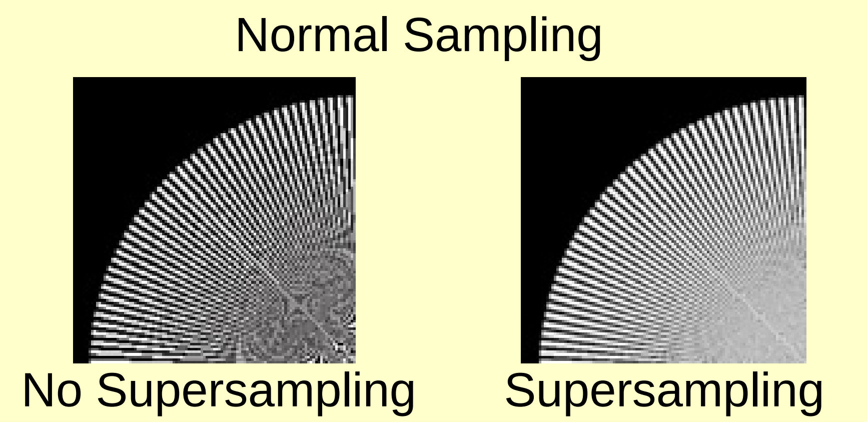
Problem: no supersampling here

Stochastic Sampling

Place sampling points randomly into pixels

- Monte Carlo method to estimate integral of shape in pixel
- Defeat artefacts in regular high-frequency patterns by making sampling irregular
- Instead of Moiré pattern: high-frequency noise ("speckle")
- Can be combined with (adaptive) supersampling and proper postfiltering

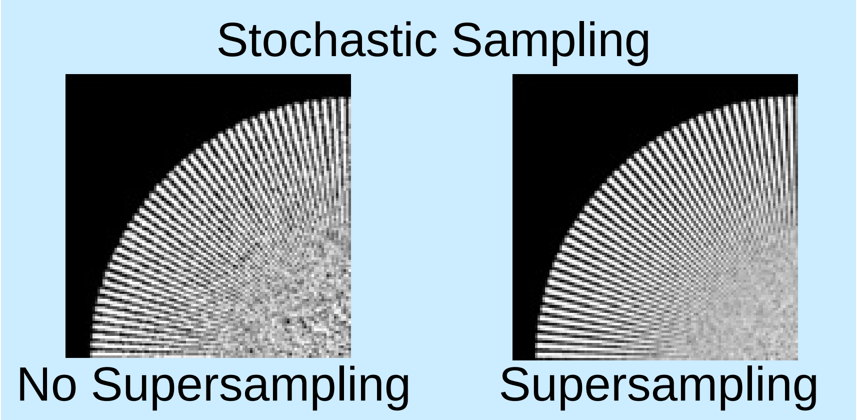




Normal Sampling

No Supersampling

Supersampling



Stochastic Sampling

No Supersampling

Supersampling

Prefiltering with OpenGL

Points and Lines

Pixel alpha values are calculated according to line/point coverage

```
glEnable(GL_LINE_SMOOTH);  // or GL_POINT_SMOOTH
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Polygons

Similar approach as as above with GL_POLYGON_SMOOTH but:

1. Disable depth buffering (because we must combine “hidden” pixels with “seen” pixels along polygon edges)
2. Sort polygons according to the depth (relative to current view position) and render them into frame buffer in front-to-back order
3. Use blending parameters GL_SRC_ALPHA_SATURATE and GL ONE (polygons that are further away cannot easily draw over closer ones)

Supersampling with OpenGL

Automatic Supersampling

```
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_MULTISAMPLE);
glEnable(GL_MULTISAMPLE);
```

Stochastic Supersampling

```
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_ACCUM); ...
glClear(GL_ACCUM_BUFFER_BIT);
for(int i = 0; i<n; i++) {  // n = number of subpixels
    // jitter camera position with random x/y in (-0.5, 0.5)
    cam.slide(jitter[i].x, jitter[i].y, 0);
    ... draw scene ...
    glAccum(GL_ACCUM, 1.0/n); // scale & add to acc. buffer
}
glAccum(GL_RETURN, 1.0);  // copy acc. buffer to screen
```

DRAWING TEXT IN OPENGL

Drawing Text to the Screen

- Need to draw a sequence of character pixmaps (bitmaps)
- Requires bitmaps for all characters of a font type and font size

Bitmap drawing function

```
void glBitmap( GLsizei width, GLsizei height, GLfloat xorig,
               GLfloat yorig, GLfloat xmove, GLfloat ymove,
               const GLubyte *bitmap )
```

Draw *width*×*height* bitmap so that bitmap pos *xorig*/*yorig* is at raster pos, then increment raster pos by *xmove*/*ymove*

GLUT text drawing functions

```
void glutBitmapCharacter( void* font, int character )
```

font given by GLUT constant; *character* code usually ASCII

```
int glutBitmapWidth( GLUTbitmapFont font, int character )
```

Returns the width of a *font*'s *character*

Drawing Text at Window Coords.

```
int textStringWrite( int xStart, int yStart, void* font,
                      float textColour[3], char* textString ) {
  // store lighting & depth test state and disable them
  glPushAttrib(GL_CURRENT_BIT | GL_LIGHTING_BIT
               | GL_DEPTH_BUFFER_BIT);
  glDisable( GL_LIGHTING ); glDisable( GL_DEPTH_TEST );
  glColor3f( textColour[0], textColour[1], textColour[2] );

  int xPos = xStart;
  glWindowPos2i( xPos, yStart );
  int numChars = strlen( textString );
  for ( int c = 0; c < numChars; c++ ) {
    glutBitmapCharacter( font , textString[c] );
    xPosn += glutBitmapWidth( font, textString[c] );
    glWindowPos2i( xPos, yStart );
  }

  glPopAttrib();  // restore state
  return xPosn;   // return next x position for convenience
}
```

Drawing Text at World Coords.

```
void textStringWrite( float x, float y, float z, void* font,
                      float textColour[3], char* textString ) {
  glPushAttrib( GL_CURRENT_BIT | GL_LIGHTING_BIT );
  glColor3f( textColour[0], textColour[1], textColour[2] );
  glDisable( GL_LIGHTING );

  // set raster position to transformed world coords.
  // then get current raster position in window coords.
  float rasterWinCoords[4];
  glRasterPos3f( x, y, z );
  glGetFloatv( GL_CURRENT_RASTER_POSITION, rasterWinCoords );

  int numChars = strlen( textString );
  for ( int c = 0; c < numChars; c++ ) {
    glutBitmapCharacter( font , textString[c] );
    rasterWinCoords[0] += glutBitmapWidth( font, textString[c] );
    glWindowPos2i( rasterWinCoords[0], rasterWinCoords[1] );
  }
  glPopAttrib();
}
```

Drawing Text with GLUT

- Example call to window coord. text drawing function:

```
textStringWrite( 50, 50, GLUT_BITMAP_HELVETICA_18,
                 myColor, "Hello World!" );
```

- Available GLUT fonts:
  - GLUT_BITMAP_8_BY_13 (8 by 13 pixel fixed width)
  - GLUT_BITMAP_9_BY_15 (9 by 15 pixel fixed width)
  - GLUT_BITMAP_TIMES_ROMAN_10 (10-point Times Roman)
  - GLUT_BITMAP_TIMES_ROMAN_24 (24-point Times Roman)
  - GLUT_BITMAP_HELVETICA_10 (10-point Helvetica)
  - GLUT_BITMAP_HELVETICA_12 (12-point Helvetica)
  - GLUT_BITMAP_HELVETICA_18 (18-point Helvetica)

Times Roman Helvetica

SUMMARY

Summary

- **Aliasing** can occur when sampling a high-frequency signal (e.g. jaggies, disappearing objects, Moiré patterns)
- Anti-aliasing can reduce aliasing artifacts
  1. **Prefiltering**: weigh object color by coverage
  2. **Postfiltering**: smooth image by averaging
  3. **Supersampling**: average over subpixels
- Drawing text in OpenGL means drawing a sequence of character pixmaps

References:

- Aliasing & Anti-Aliasing: Hill, Chapter 9.8
- OpenGL API Reference: http://www.cs.auckland.ac.nz/compsci372s1c/resources/manpagesOpenGL

Quiz

1. What is aliasing?
2. Describe three typical aliasing artifacts.
3. How does prefiltering work?
4. How does stochastic supersampling work?