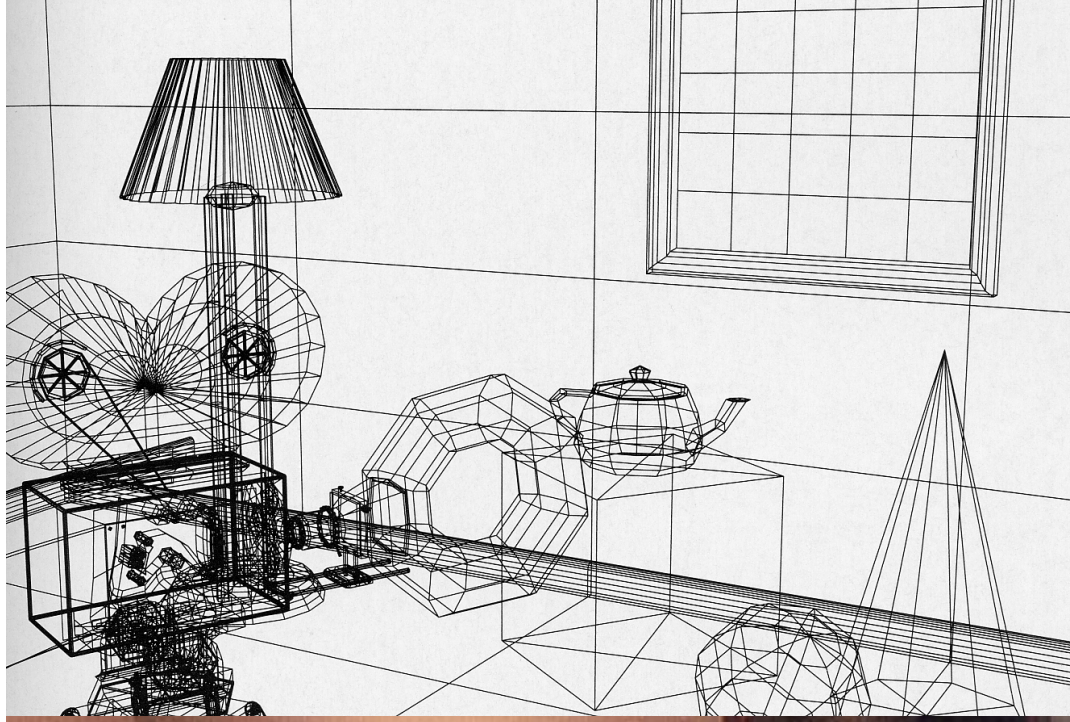
Attributes of Graphics Primitives Primitives



© 2005 James K. Ha

A note on what will be on the exams… and OpenGL

- Concepts
- No OpenGL syntax questions
- No coding…pseudo code
- I will skip details of syntax…better to read it on your own
- I will cover concepts and features of OpenGL
- Use standard linear algebra operators - dot product, cross product, matrix multiplication

© 2005 James K. Hahn 2010 Robert Falk

Some Additional Resources

www.opengl.org : All things OpenGL

www.OpenGL.org/documentation/specs : The formal OpenGL specification

www.opengl.org/documentation/books : Opengl Books
The OpenGL Programming Guide: Version 3.0 and 3.1
The OpenGL Shading Language 2nd edition
OpenGL programming on Mac OS X

www.glprogramming.com/red : The Original OpenGL redbook in HTML

http://chortle.ccsu.edu/VectorLessons/VectorIndex.html : 3d Math tutorial

© 2005 James K. Hahn 2010 Robert Falk

Attribute parameters

- Parameter that affect the way a primitive is to be displayed
  - Color, size, texture
- Possible approaches:
  - Extend vertex parameter list to include all parameters, e.g:

```
glVertex3f(float x, float y, float z, float r, float g, float
           b,...); //Bad Idea
```

  - Maintain list of current attribute values (state variables or parameters)

```
glColor3f(float r, float g, float b); //Set color (size, etc) first
glVertex3f(float x, float y, float z);//Then draw (Better Idea)
```

© 2005 James K. Hahn 2010 Robert Falk

OpenGL State Variables

- Examples:
  - Color, texture
  - Matrix mode
  - Model-view matrix
  - Lighting effects
- Primitives use attributes in the current state list
- Changes in attributes affect primitives that are specified after the state change

© 2005 James K. Hahn 2010 Robert Falk

Color

- Display mode (at program start)
  - Example

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
```

  - First argument specify single buffer (or double buffer)
  - Second argument specify each pixel will have values of R, G, B (or RBGA)
- Set current color using

```
glVertex3f( 1.0, 3.0, 2.0); //color??
glColor3f( 1.0, 0.0, 1.0 );
glVertex3f( 5.0, 2.0, 9.0); //purple!
```

© 2005 James K. Hahn 2010 Robert Falk

Color Blending
Image Composition

- Useful for transparency, anti-aliasing, composition, etc.
- Draw object to frame buffer (frame buffer is destination color)
- Combine new object with source color (R_s, G_s, B_s, A_s) with frame buffer (destination color) (R_d, G_d, B_d, A_d) and store in frame buffer:
  - $(R_d, G_d, B_d, A_d) =$
    $(s_r R_s + d_r R_d,\ s_g G_s + s_g G_d,\ s_b B_s + d_b B_d,\ s_a A_s + d_a A_d)$
    Sum of source and destination values weighted by blending factors
  - Blending factors
    Source: (s_r, s_g, s_b, s_a) - default is (1,1,1,1)
    Destination: (d_r, d_g, d_b, d_a) - default is (0,0,0,0)
  - OpenGL glBlendFunc

© 2005 James K. Hahn 2010 Robert Falk

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

glBegin(GL_QUADS);
    glColor4f(1.0f, 0.0f, 0.0f, 1.0f);
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(1.0f, 0.0f, 0.0f);
    glVertex3f(1.0f, 0.2f, 0.0f);
    glVertex3f(0.0f, 0.2f, 0.0f);

    glColor4f(0.0f, 1.0f, 0.0f, 0.6f);
    glVertex3f(0.8f, 0.0f, 0.0f);
    glVertex3f(1.0f, 0.0f, 0.0f);
    glVertex3f(1.0f, 1.0f, 0.0f);
    glVertex3f(0.8f, 1.0f, 0.0f);

    glColor4f(0.0f, 0.0f, 1.0f, 0.4f);
    glVertex3f(0.0f, 0.8f, 0.0f);
    glVertex3f(1.0f, 0.8f, 0.0f);
    glVertex3f(1.0f, 1.0f, 0.0f);
    glVertex3f(0.0f, 1.0f, 0.0f);

    glColor4f(1.0f, 0.0f, 1.0f, 0.2f);
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(0.2f, 0.0f, 0.0f);
    glVertex3f(0.2f, 1.0f, 0.0f);
    glVertex3f(0.0f, 1.0f, 0.0f);
glEnd();
```

$$C_d = A_s * C_s + (1.0 - A_s) * C_d$$

© 2005 James K. Hahn 2010 Robert Falk

Attributes

- All Primitives
  - Color: `glColor4f(GLfloat r,GLfloat g,GLfloat b,GLfloat a)`
- Point Attributtes
  - Size: `glPointSize(GLfloat size)`
- Line Attributes
  - Style: `glLineStipple(GLint factor, GLushort pattern)`
  - Width: `glLineWidth(GLfloat width)`
- Fill Area Attributes
  - Style: `glPolygonStipple(GLubyte *pattern)`
  - Mode: `glPolygonMode(face,GL_{POINT,LINE,FILL})`
  - Fill: `glShadeModel(GL_{SMOOTH, FLAT})`
  - Texture: Chapter 18
  - face: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK

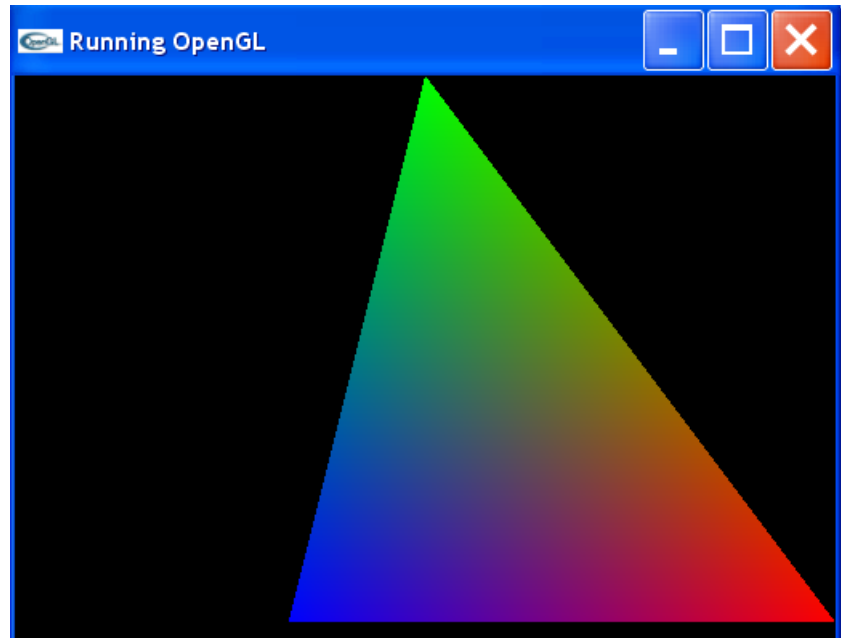
© 2005 James K. Hahn, 2010 Robert Falk

Smooth Fill

- Interpolate vertex values (more realistic smooth appearance)
- Example: interpolate vertex color of triangle
- GL_FLAT: whole primitive takes color of 1 vertex

```
glShadeModel( GL_SMOOTH );

glBegin ( GL_TRIANGLES );
  glColor3f (0.0, 0.0, 1.0);
  glVertex2i( 50, 50 );
  glColor3f (1.0, 0.0, 0.0);
  glVertex2i( 150, 50 );
  glColor3f (0.0, 1.0, 0.0);
  glVertex2i( 75, 150 );
glEnd();
```

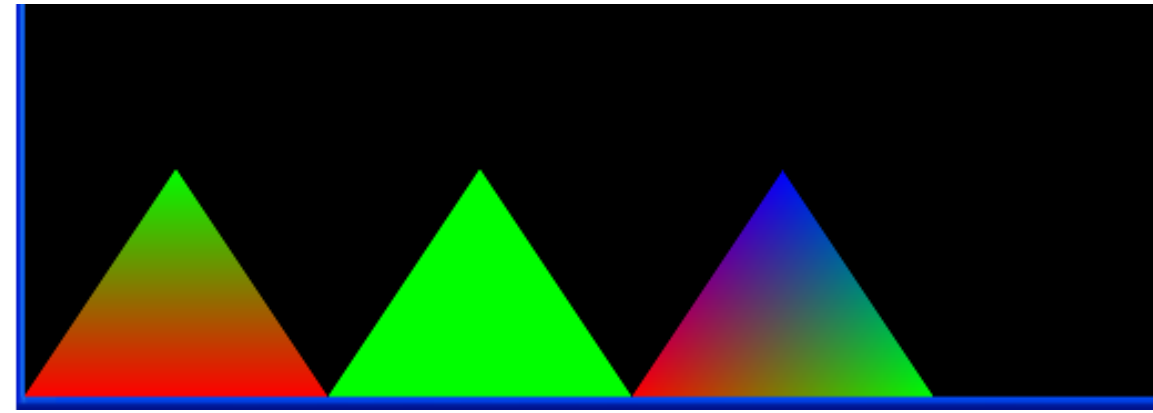


© 2005 James K. Hahn 2010 Robert Falk

```
gluOrtho2D(0.0f, 150.0f, 0.0f, 150.0f);
 glColor3f(1.0f, 0.0f, 0.0f);
glBegin(GL_TRIANGLES);
    glVertex2f(0.0f, 0.0f);
    glVertex2f(40.0f, 0.0f);
    glColor3f(0.0, 1.0f, 0.0f);
    glVertex2f(20.0f, 20.0f);
glEnd();

glBegin(GL_TRIANGLES);
    glVertex2f(40.0f, 0.0f);
    glVertex2f(80.0f, 0.0f);
    glVertex2f(60.0f, 20.0f);
glEnd();

glBegin(GL_TRIANGLES);
    glColor3f(1.0, 0.0f, 0.0f);
    glVertex2f(80.0f, 0.0f);
    glColor3f(0.0, 1.0f, 0.0f);
    glVertex2f(120.0f, 0.0f);
    glColor3f(0.0, 0.0f, 1.0f);
    glVertex2f(100.0f, 20.0f);
glEnd();
```

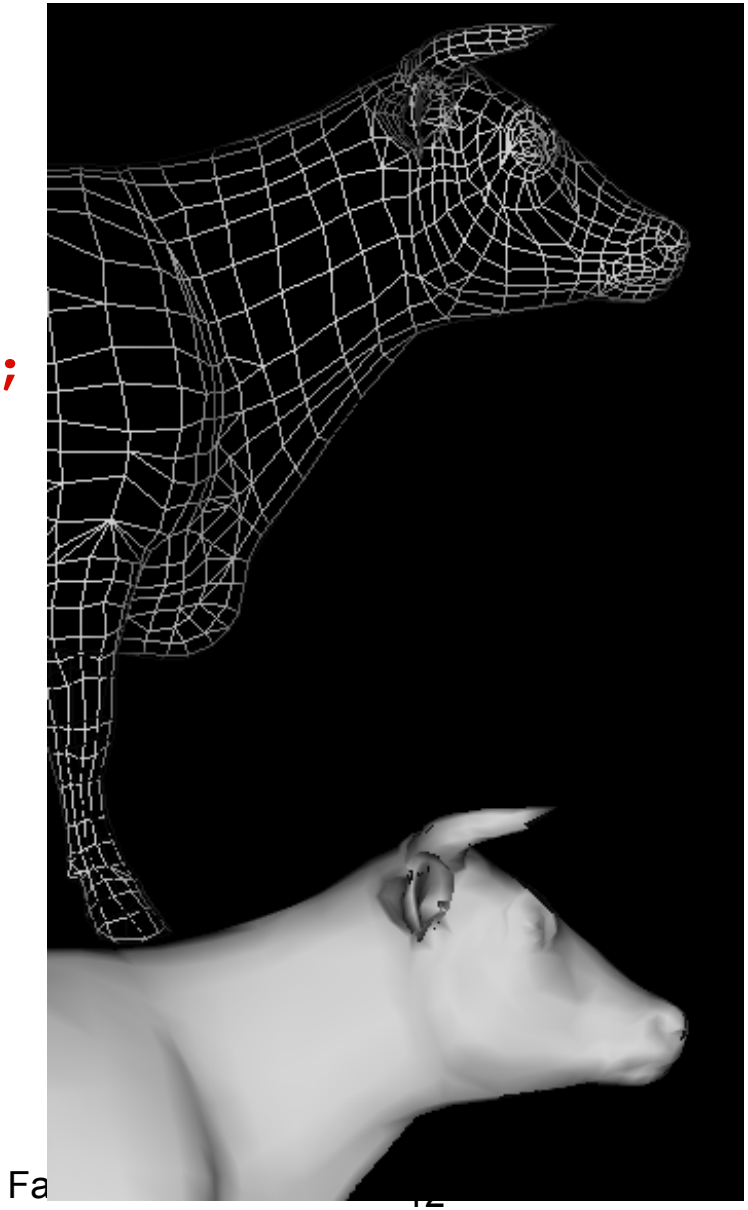


©2005 James K. Hahn 2010 Robert Falk

OpenGL Polygon Fill

Line Mode:

```
glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
```



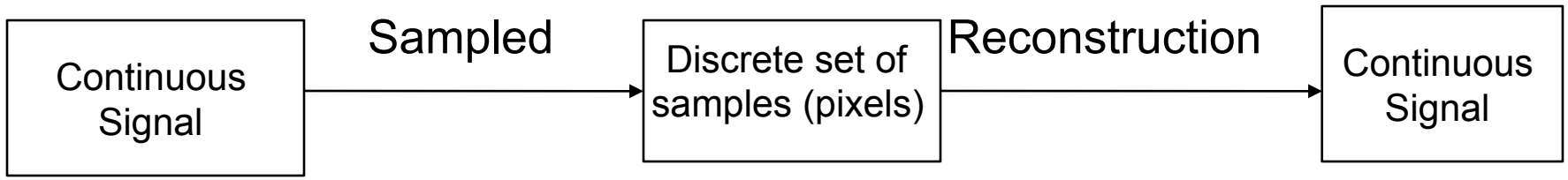
Fill Mode:

```
glPolygonMode(GL_FRONT,GL_FILL);
```

Can also fill with texture data (chap 18)

© 2005 James K. Hahn 2010 Robert Fa

Basic Problem with Digital Representation

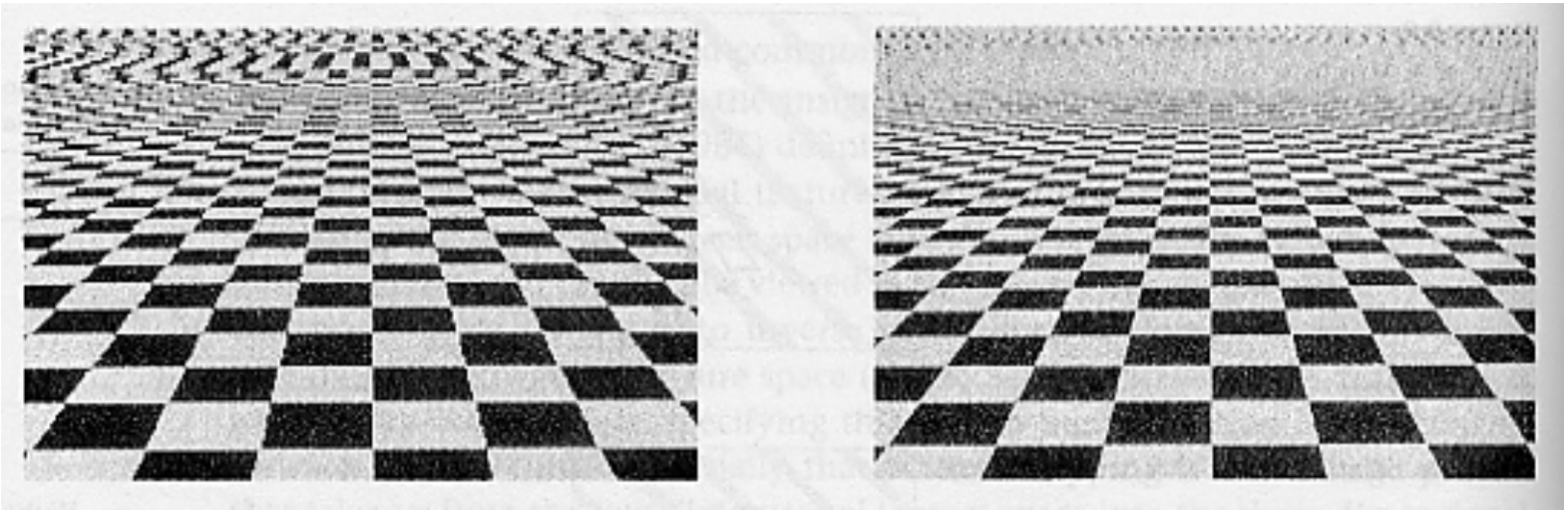


– How to sample and reconstruct so that we end up with what we started with?

© 2005 James K. Hahn 2010 Robert Falk

Anti-aliasing

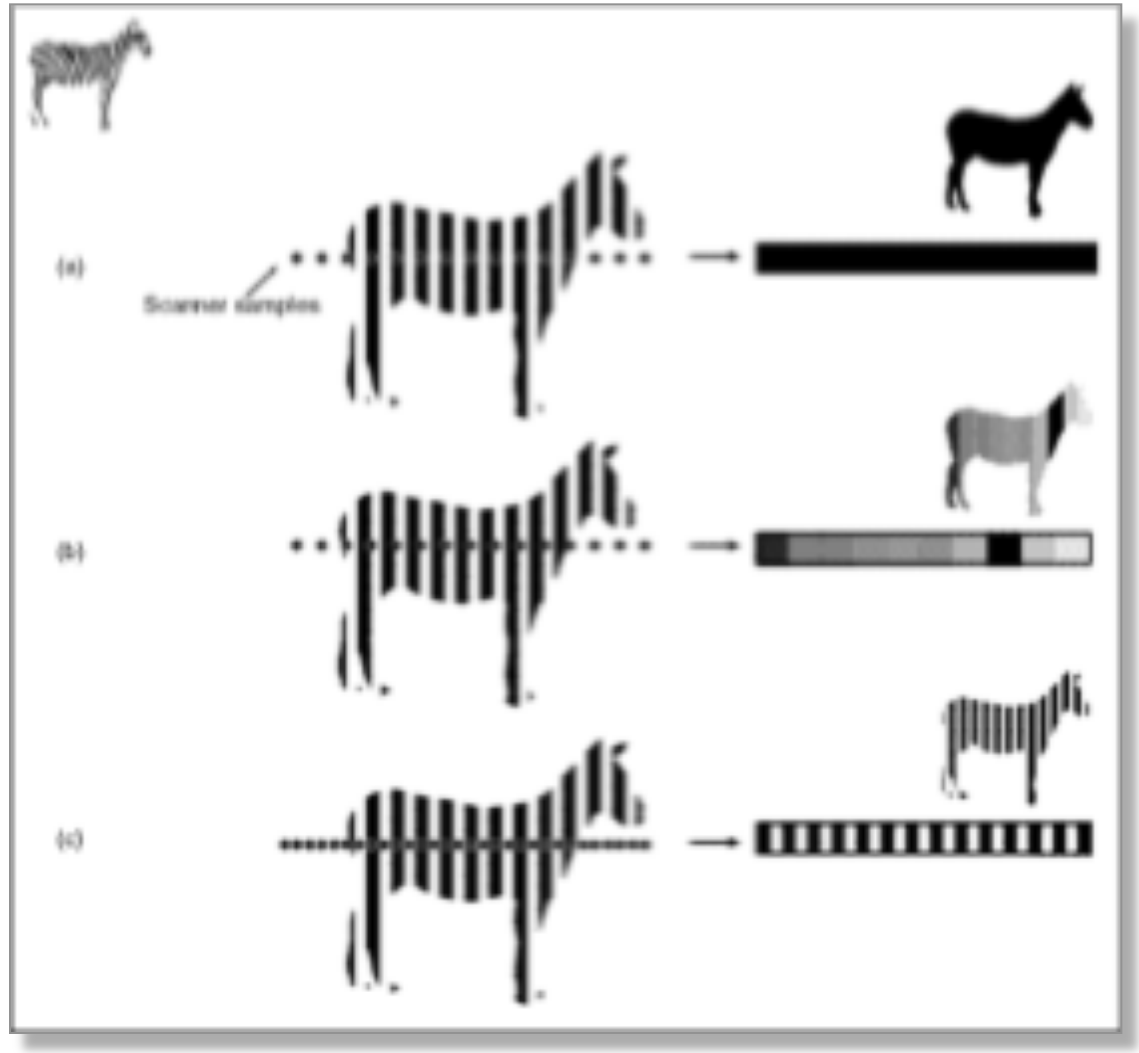
- Right image is “anti-aliased” reducing but not eliminating the artifact
- Aliasing has moved to a higher frequency



© 2005 James K. Hahn 2010 Robert Falk

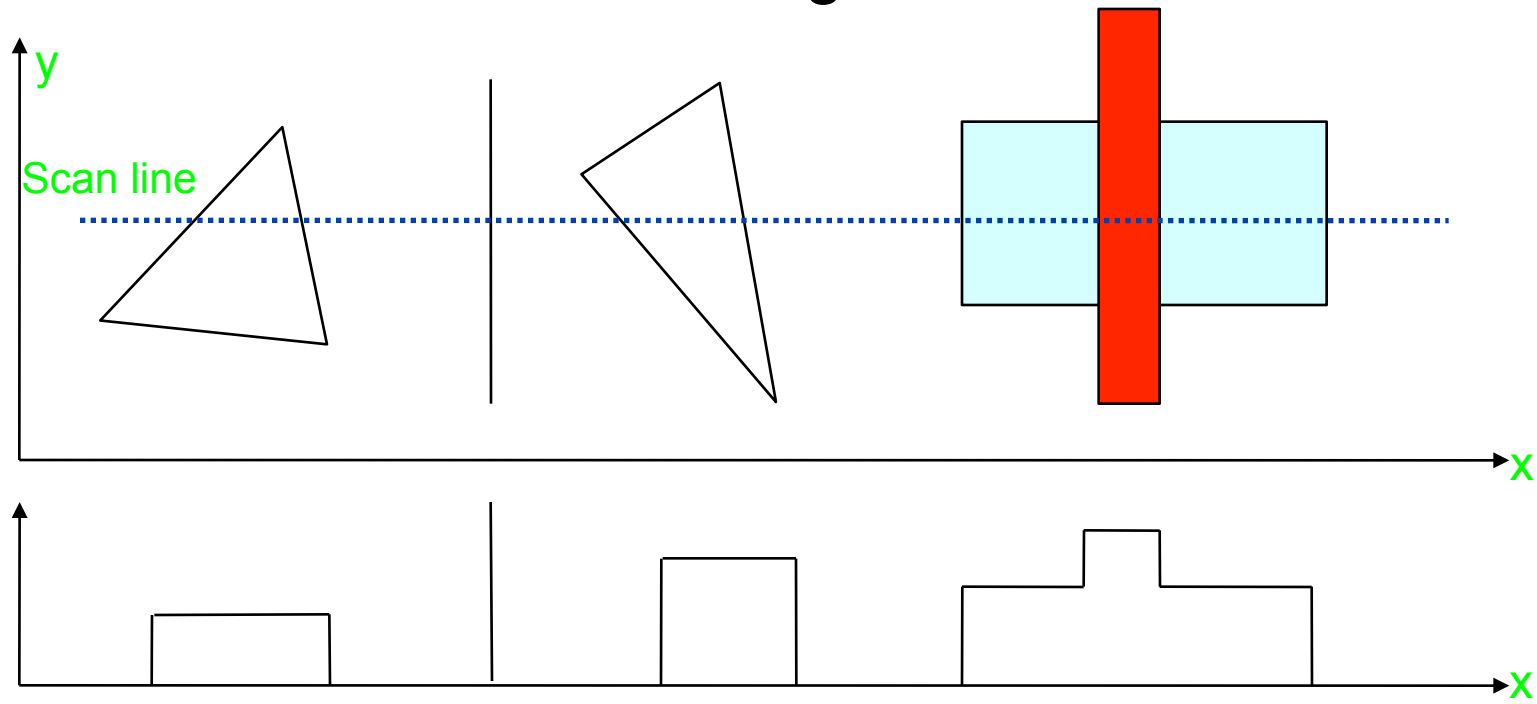
Cause of Aliasing

- Aliasing is the presence of low frequency signals where there are no such signals
- With insufficient sampling rate, the high-frequency signals masquerade as low frequency signals
- In example, a and b show aliases of the original signal



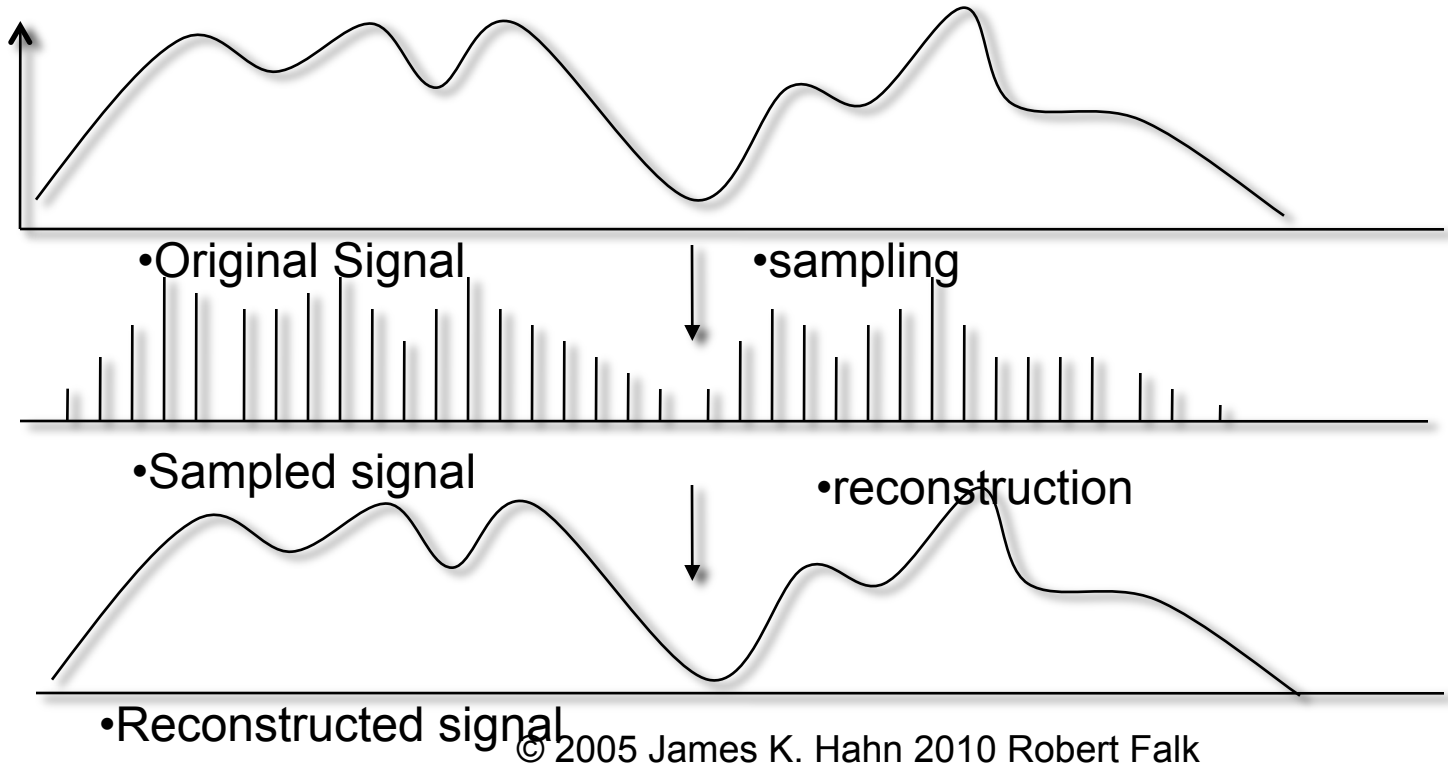
© 2005 James K. Hahn 2010 Robert Falk

– Before scan-conversion, projection of 3-D objects onto viewplane can be thought of as a continuous 2-D signal



© 2005 James K. Hahn 2010 Robert Falk

– Goal is to reconstruct a signal which is as close as possible to the original sample signal

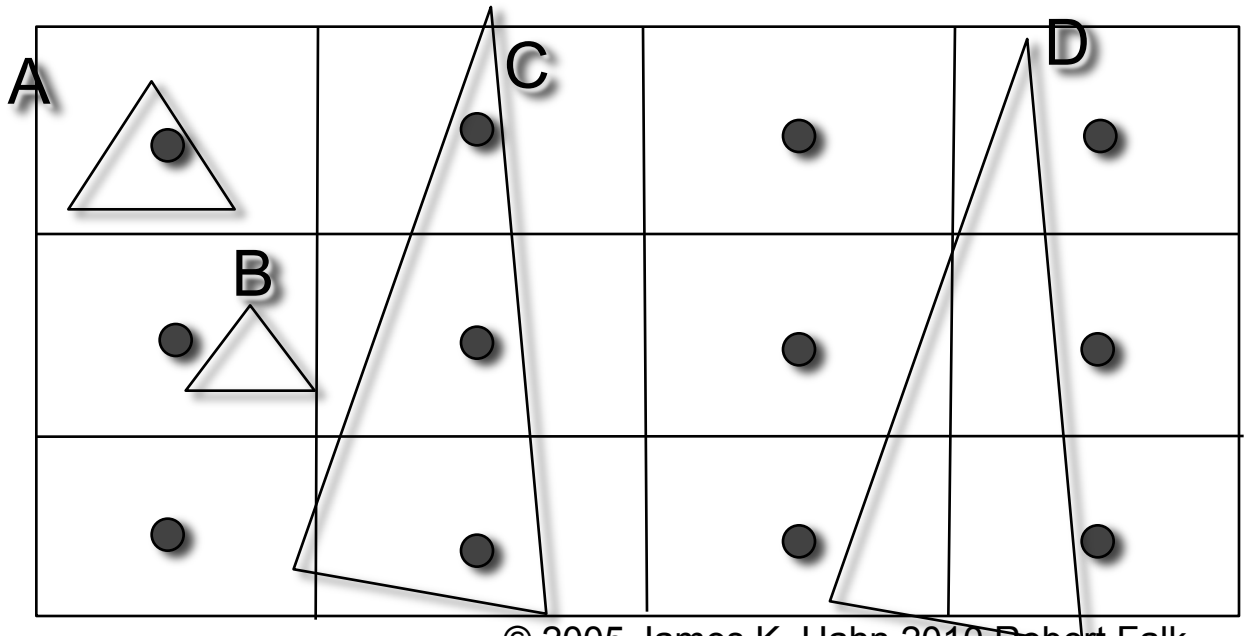


© 2005 James K. Hahn 2010 Robert Falk

Point Sampling

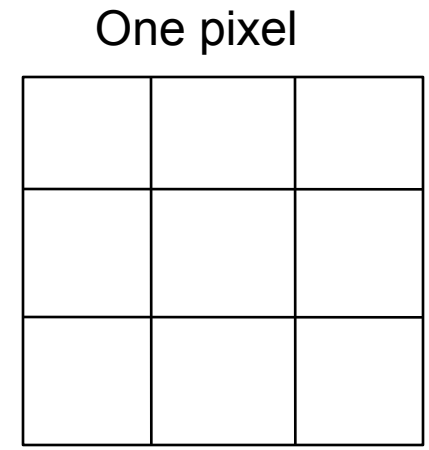
– Select one point for each pixel; evaluate the original signal at this point; assign value to the pixel

– Important features of the sample may be missed:



© 2005 James K. Hahn 2010 Robert Falk

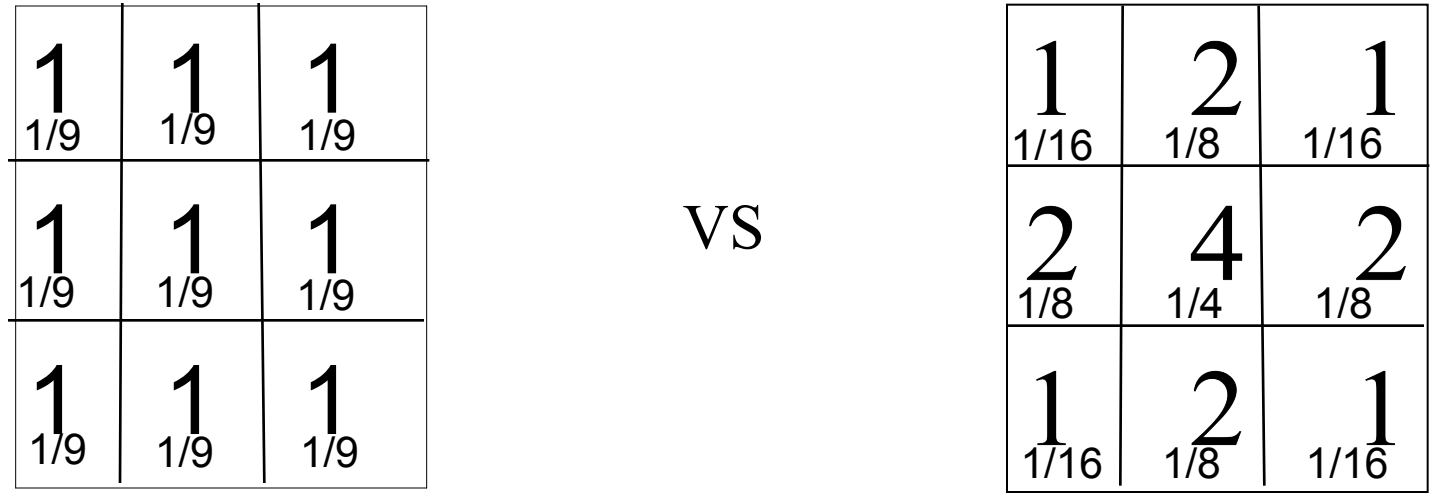
Supersampling



- Increase sampling rate
- Take multiple adjacent samples and average their values to determine a value for a single pixel
- Popular in graphics; easy and often achieves good results despite increase in computation
- However, sometimes minimum sampling rate must be arbitrarily high!

© 2005 James K. Hahn 2010 Robert Falk

– Uniform vs. non-uniform (Bartlett window) weighting
  - Intuitively, the center sample has more importance than side samples
  - Samples sum to 1.0



© 2005 James K. Hahn 2010 Robert Falk

Shannon's theorem

- Signal can be properly reconstructed from its samples if original signal is sampled at frequency that is at least twice the highest frequency component in its spectrum
- Lower bound on sample rate known as the Nyquist frequency
- Sampling below the Nyquist frequency can produce what could have been obtained from sampling a lower frequency signal
- The low frequency is an alias of the high frequency signals

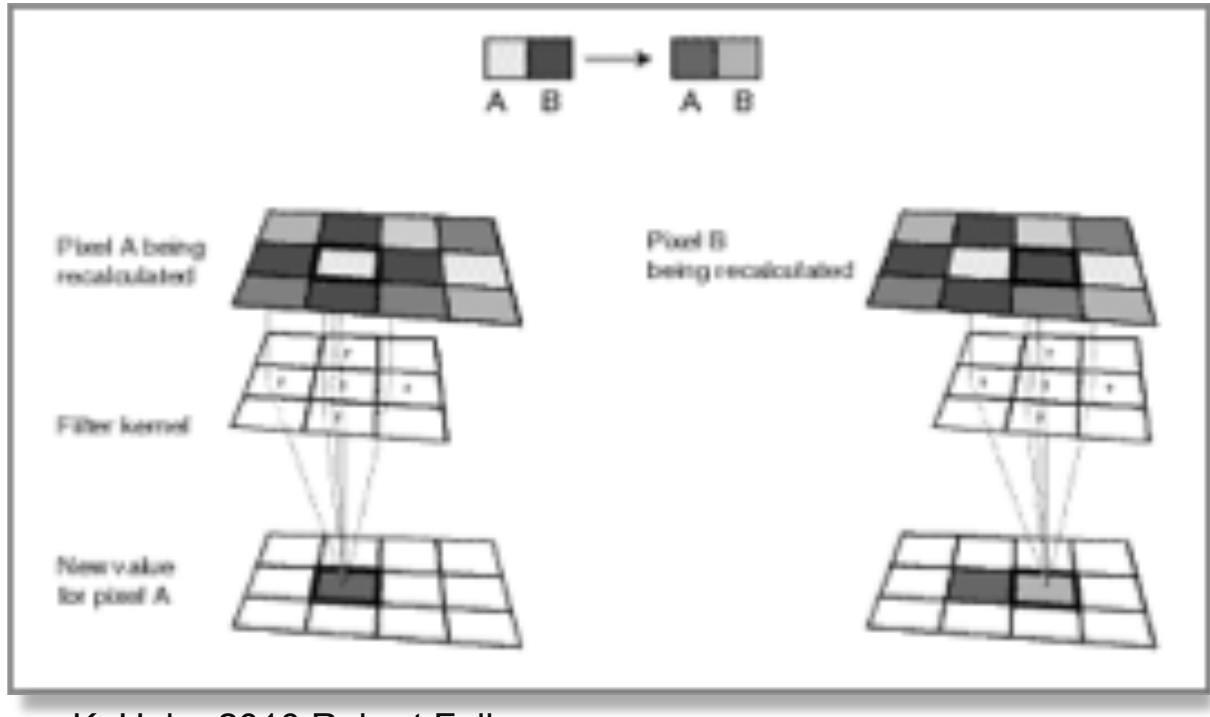
© 2005 James K. Hahn 2010 Robert Falk

Filtering

- Previous problem was high-frequency components masquerading as low-frequency components in the reconstructed signal
- Solution : remove the high frequencies from the original signal
  - this is known as band-limiting or low-pass filtering
  - tradeoff aliasing vs blurring
- Intuitively
  - high detail have high frequency components
  - sharp edges have high frequency components!

© 2005 James K. Hahn 2010 Robert Falk

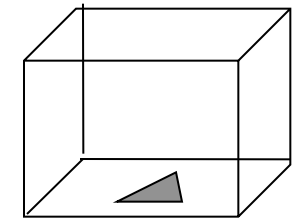
- Filtering performed by averaging the signal over an area (the kernel of the filter)
- Intuitively, detail is lost
- For more rigorous explanation: CS6554

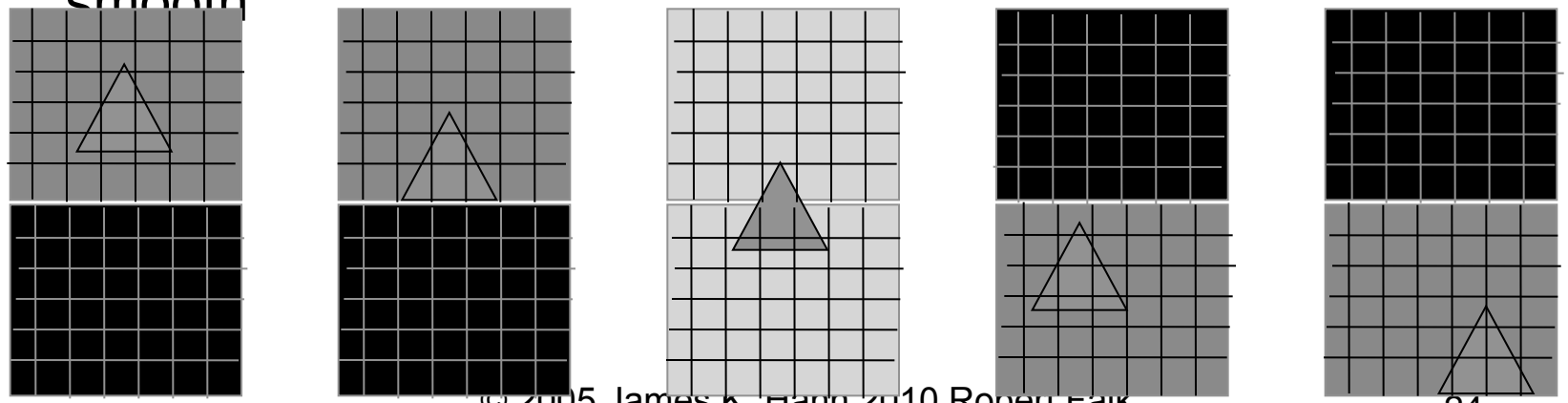


© 2005 James K. Hahn 2010 Robert Falk

Uniform Weighted Area Sampling

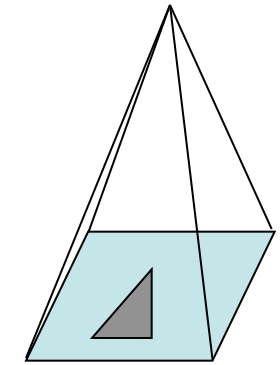
- Instead of point sample, integrate the signal over the area of a square centered about each grid point and selecting the average intensity as that of the pixel
- Often called a box-filter
- No regard to location of object in pixel
- Movement from one pixel to another may not appear smooth



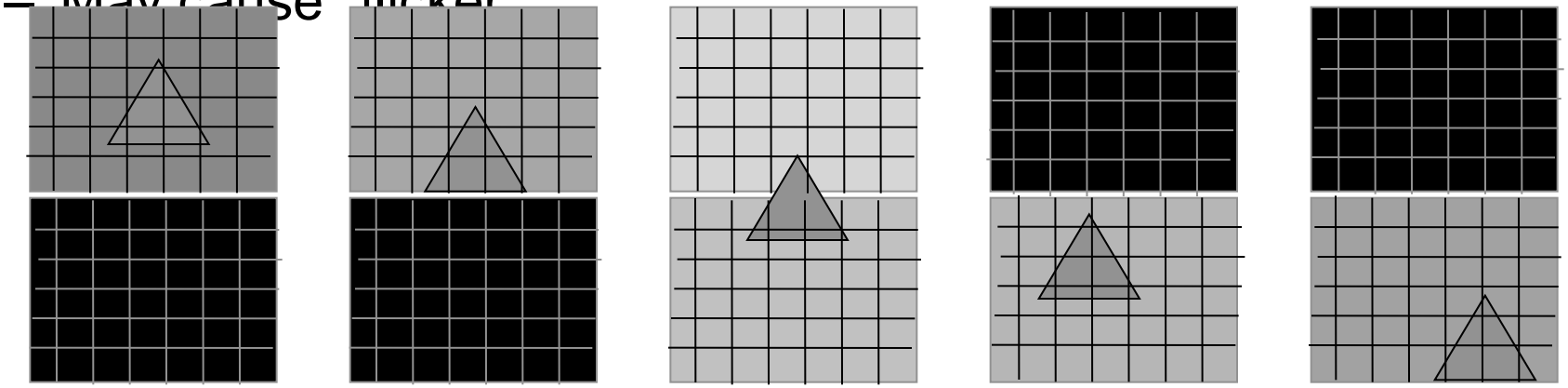


© 2005 James K. Hahn 2010 Robert Falk

Non-Uniform Weighted Area sampling



- Assign different weights to different parts of the pixel
  - For example using a pyramid or cone-shaped “filter”
- Object still only contributes only to the pixel containing it
- May cause “flicker”



- Fix this by allowing weighting functions to overlap

© 2005 James K. Hahn 2010 Robert Falk

Antialiasing and the “Jaggies”

- “Jaggies” are NOT aliases of high-frequency signal masquerading as low frequency signal!
- The high frequencies are still there but in a different direction (in x and y direction as opposed to direction of underlying geometry)

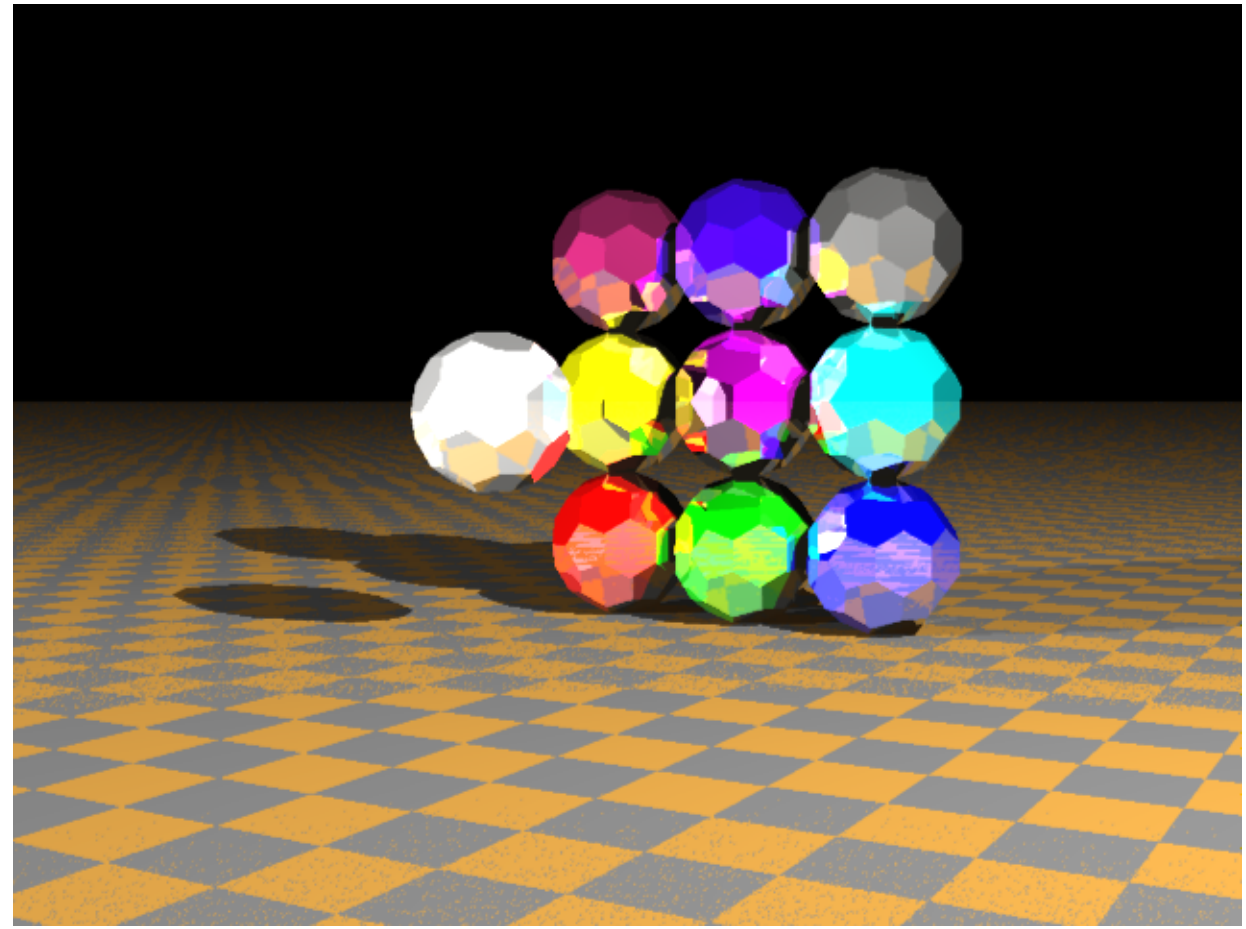
© 2005 James K. Hahn 2010 Robert Falk

Filtering, Aliasing and Jaggies

- Filtering reduces high frequency components
- Filtering also reduces jaggies, which causes confusion
- “Jaggies” are not examples of aliasing artifacts!

© 2005 James K. Hahn 2010 Robert Falk

Next Topic:
Geometric Transformations



© 2005 James K. Hahn 2010 Robert Falk