Attributes and Animation

- Pattern filling in scan-conversion
- Antialiasing
- Double-buffering and animation
ATTRIBUTES

• Any parameter that affects a primitive displayed is an attribute parameter.

• Polygon Attributes
  – pattern, color, anti-aliasing

• Line Attributes
  – type (solid, dashed, dotted), width, cap (butt, round, square), join (miter, round, etc.), pattern, color, anti-aliasing
**PATTERN FILLING (bitmap patterns)**

- The 1s & 0s select foreground and background color, respectively. For transparent mode, only 1s are written with foreground color.

- Anchor the pattern at a vertex of a primitive; this choice allows the pattern to move with the primitive.

- Anchor the pattern at the corner of the drawing area; the pattern does not move with the primitive.

- Generating patterns while scan-converting, or scan converting a primitive first into a rectangular work area, and then copy pixels from the rectangular bitmap to the appropriate space in the background.
Triangle Pattern

• Generating color patterns on a span (horizontal line)

```java
void span(int x2, int x1, int y) {
    for (int x = x1; x<x2; x++) {
        if (cnt % 100 < 50) // triangle pattern anchored on frame
            gl.glColor3d(Math.sin(x/5), Math.cos(x/5), 1);
        else // triangle pattern anchored on triangle coordinates
            gl.glColor3d(Math.sin((x-x1)/5), Math.cos((x-x1)/5), 1);
        drawPoint(x, y);
    }
}
```

Copyright © 2002 by Jim X. Chen:

jchen@cs.gmu.edu
Point and Line Attributes

- void glPointSize(GLfloat size);
- void glLineWidth(GLfloat width);
- glLineStipple(1, 0x3F07);
  glEnable(GL_LINE_STIPPLE);
- void glLineStipple(GLint factor, Glushort pattern);

Stippled Lines
SPECIFYING A COLOR

```c
#include <GL/gl.h>

void draw_object(A)
{
    glColor3f(0.0, 0.0, 0.0);
    draw_object(A);
    glColor3f(1.0, 0.0, 0.0);
    draw_object(B);
    glColor3f(0.0, 1.0, 0.0);
    draw_object(C);
}
```

black
red
green
yellow
blue
magenta
cyan
white

Copyright @ 2002 by Jim X. Chen:

jchen@cs.gmu.edu
Line Anti-aliasing

Antialiasing Demo
Supersampling (postfiltering)

• Increase sampling rate, each pixel area is covered by a finer grid

We can then use multiple sample points in a pixel that is within a line to determine an appropriate intensity level for the pixel.

Supersampling algorithms often choose more weight to subpixels near the center of a pixel area: pixel-weighting masks

\[
\begin{bmatrix}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1 \\
\end{bmatrix}
\]
Area sampling (prefiltering)

Determine pixel intensity by calculating the areas of overlap of each pixel with the objects to be displayed

- **Unweighted Area Sampling**
  
  determines the pixel intensity by the overlap area only

- **Weighted Area Sampling**

  an area closer to the pixel’s center has greater influence on the pixel’s intensity than an equal area further away from the pixel’s center.
FILTER FUNCTIONS

- Cone filter, Gaussian filter, Intensity table, etc.
- A simple equation for 3-pixel wide line
  - Given the current pixel’s color \((r, g, b)\), we can modify the intensity by: \((r, g, b)*(1-D/1.5)\). Therefore, the pixels have different intensity levels depending on their distances from the center of the line.
  - For a foreground color \(F\) and background color \(B\):
    \[ F*(1-D/1.5) + B*D/1.5. \]
  - We can read the current pixel color from the framebuffer.
    \[ \text{gl.glReadPixels}(x, y, 1, 1, \text{GL.GL_RGB, GL.GL_UNSIGNED_BYTE, buffer}); \]
- How to calculate \(D\)?
Antialiasing a straight line

\[ D_{NE} = D_E - \cos \alpha \]
\[ = D + \sin \alpha - \cos \alpha \]
void IntensifyPixel(int x, int y, double D)
{
    float d, r1, g1, b1;

    if (D<0) d = -D;
    else d = D;

    r1=r*(1-d/1.5); g1=g*(1-d/1.5); b1=b*(1-d/1.5);
glColor3f(r1, g1, b1);
    writepixel(x, y);
}

**Example: J1_4_Line**

**Example: J1_4_Line_Background**
void antialiasedLine(int x0, int y0, int xn, int yn) {
    int dx, dy, incrE, incrNE, d, x, y;
    float D=0, sin_a, cos_a, smc_a, Denom;

dy=yn-y0; dx=xn-x0; x=x0; y=y0; d=2*dy-dx;
    incrE=2*dy; incrNE=2*(dy-dx);

    Denom = sqrt(dx*dx + dy*dy);
    sin_a = dy / Denom; cos_a = dx / Denom;
    smc_a = sin_a - cos_a;

    while (x<xn+1) {
        IntensifyPixel(x,y,D); // current pixel
        IntensifyPixel(x,y+1,D-cos_a); // North
        IntensifyPixel(x,y-1,D+cos_a); // South

        x++;
        if (d<=0) { D+=sin_a; d+=incrE; }
        else { D+=smc_a; y++; d+=incrNE; }
    }
} /* AntiAliased Midpoint Algorithm */
Frame Rate and Refresh Rate

• If the display refreshes at 60 times per second (f/s), this means that the fastest image frame rate you can achieve is 60 f/s

• What often happens is that the frame is too complicated to draw in 1/60 second, so each frame is displayed more than once

• Notice the difference between the image frame rate and display refresh rate.
**Drawing a circle by subdivision**

- **depth=0**, draw 4 triangles. **depth=1**, each triangle is subdivided into two and we draw 8 triangles.

- consider \( v_1 \), \( v_2 \), and \( v_{12} \) as vectors. Then, \( v_{12} \) is in the direction of \( (v_1 + v_2) = (v_{1x} + v_{2x}, v_{1y} + v_{2y}, v_{1z} + v_{2z}) \) and \(|v_1| = |v_2| = |v_{12}|\).

- \( v_{12} = \text{normalize}(v_1 + v_2) \). Normalizing a vector is equivalent to scaling the vector to a unit vector.

- recursively subdivides **depth** times and draws \( 2^{depth} \) triangles.
Example: J1_5_Circle

subdivideCircle(int radius, float[] v1, float[] v2, int depth) {
    float v11[] = new float[3]; float v22[] = new float[3];
    float v00[] = { 0, 0, 0 }; float v12[] = new float[3];
    if (depth == 0) {
        for (int i = 0; i < 3; i++) {
            v11[i] = v1[i] * radius; v22[i] = v2[i] * radius;
        }
        drawtriangle(v11, v22, v00);
        return;
    }
    normalize(v12);

    // subdivide a triangle recursively, and draw them
    subdivideCircle(radius, v1, v12, depth - 1);
    subdivideCircle(radius, v12, v2, depth - 1);
}
HW1_8_Circle

- Draw a pentagon with antialiasing that rotates in a circle. Within the star there is a filled circle. The filled circle should be filled through subdivision with your own triangle function.
- The triangle should have a pattern of your choice. For example, random generated color. Clipping should be still in force.
HW3: 2013 Fall Class

1. Modify from previous homework, so that the points become small circles that bounce; (40%)
2. The small circles bounce with one another as well; (40%)
3. The circles are not supposed to penetrate one another; (20%)