Katarina Mitchell, James Rich and Michael Bullis
CprE 480x
Final Project

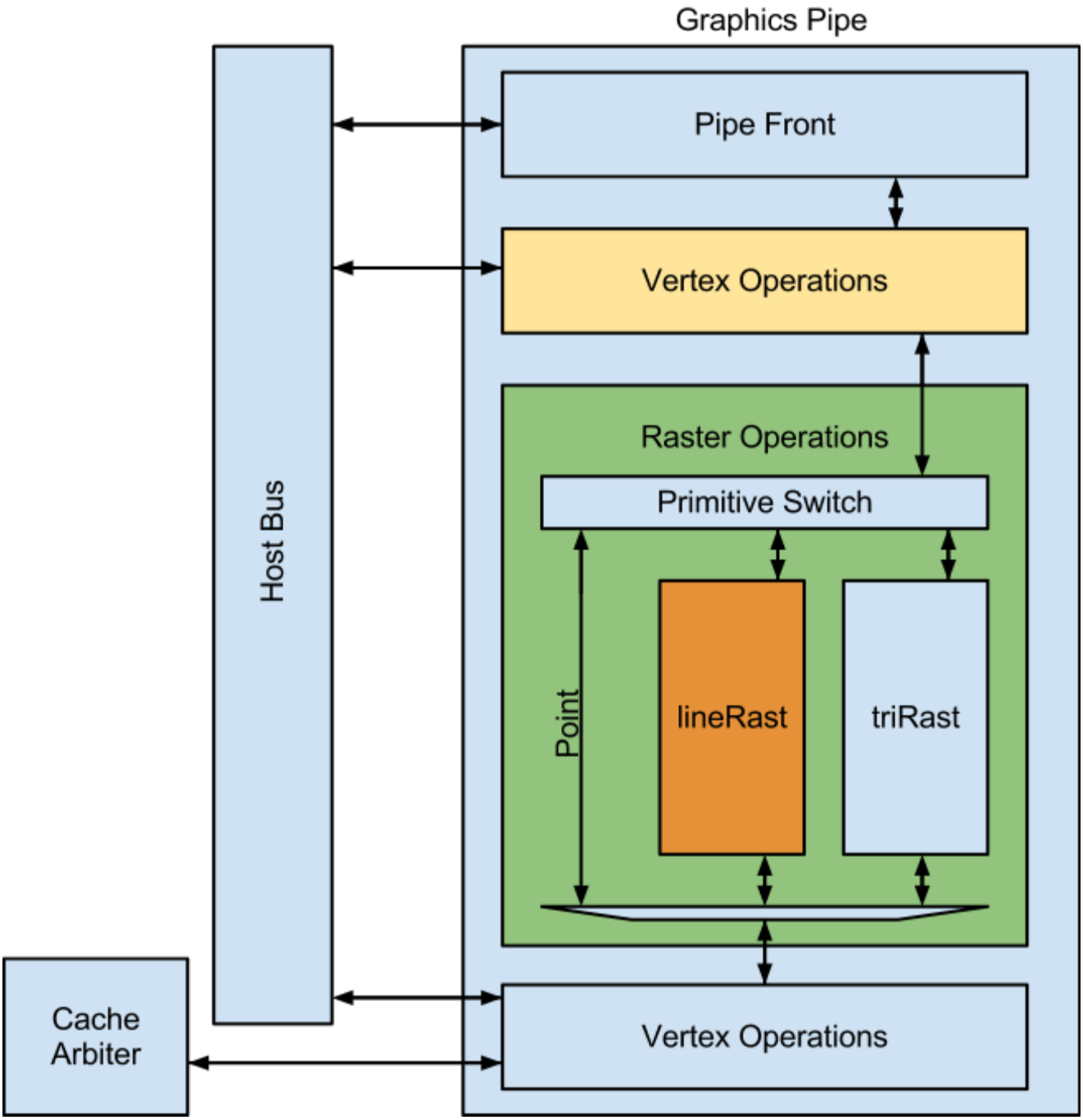
Line Rasterization and Antialiasing

Rubric

Attributes	Unsatisfactory	Satisfactory	Beyond Satisfactory
Hardware Line Rasterization & Aliasing	**Rasterize an aliased line (Bresenham) [50]**	**Use algorithm for antialiasing (Gupta-Sproull) [100]**	Further optimize the Gupta-Sproull algorithm implementation [150]
OpenGL Integration	**Basic GL_LINES functionality implemented. [25]**	**Support for GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP [50]**	Support for line color interpolation and glLineWidth [75]
Demo and Report	**Demo and report partially completed [25]**	**Full demo, report includes description of all major components [50]**	**Full demo, report also includes detailed figures and testing results [75]**

Code Modified and Added

1. OpenGL
    a. GL_LINES, GL_LINE_LOOP & GL_LINE_STRIP all implemented in original code as well as Enable/Disable antialiasing
        i. utils/src/simpleGL/glwrapper
        ii. utils/src/simpleGL/gl.h (gl_lines, gl_line_loop, gl_line_strip are opcodes x0001-3)
        iii. GL_LINE_SMOOTH within glEnable/glDisable (GL_LINE_SMOOTH opcode x0B20)
2. Assembly
    a. GL_LINES, GL_LINE_LOOP & GL_LINE_STRIP all handled in original code
3. VertexOps
    a. Added logic to the clipping component to remove lines outside of the viewport
4. RasterOps
    a. Added the LineRast component and necessary signals.
    b. Added a link between the host bus and the LineRast component.
5. LineRast
    a. Added two separate algorithms, Bresenham Line Rasterization algorithm and Gupta-Sproull Antialiasing Line Rasterization Algorithm.



OpenGL Additions

New types in SGP_config

In utils/src/simpleGL/glwrapper.c we had to modify the *glEnable* and *glDisable* methods to send a packet to the *rasterOps* module whenever a *glEnable(GL_LINE_SMOOTH)* or *glDisable(GL_LINE_SMOOTH)* call is made. The *rasterOps* module will then receive this with its host bus interface and forward this onto our line rasterization unit.

```
type line_array_t is array(1 downto 0) of vertexAttr_t;

type line_t is
        record
                vertex    :   line_array_t;
                valid     :   std_logic;
                smooth :   std_logic;
        end record;
```

In utils/include/sgp.h

```
#define SMOOTH_ENABLE    0x400
#define SMOOTH_DISABLE   0x800
```

In utils/src/simpleGL/glwrapper.c

glEnable

```
        if(cap == GL_LINE_SMOOTH) {
                sendPacket( RASTEROPS | SMOOTH_ENABLE | NUMPACKETS(0),0);
        }
```

glDisable

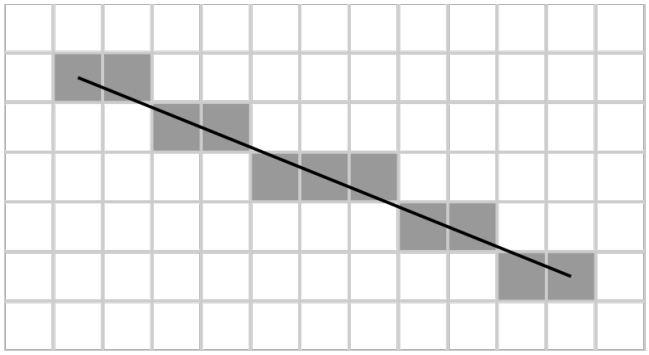
```
        if(cap == GL_LINE_SMOOTH) {
                sendPacket( RASTEROPS | SMOOTH_DISABLE | NUMPACKETS(0),0);
        }
```

Line Rasterization using Bresenham Algorithm

For the case when antialiasing is disabled, the lineRast module uses a variation of the Bresenham algorithm to rasterize lines. The algorithm is fairly fast and requires only one division per line to be rasterized. All other calculations are simple additions or subtractions.

The basic idea of the Bresenham algorithm is to start at the left side of the line to be rasterized and increment x as you continue on. The y direction is only changed when the error (or slope) indicates.

The algorithm starts by reading in a line, identifying the slope and swapping the x-values with the y-values if the slope is too steep (if $\Delta y > \Delta x$). The reason we need to swap the x-values with the y-values for a line with a steep slope is because we are using $\Delta y/\Delta x$ to calculate our error. This is the distance the rasterized fragment will move away from the true line during each iteration when we increment our x-value. If the error gets too large, then it means that the y-value is too far away from the true line we are trying to draw. When the total error exceeds 0.5, the y-value is incremented or decremented by 1 (depending on orientation) and 1 is subtracted from the total error. If we do not swap the x-values and y-values for a vertical line, then our deltaE ($\Delta y/\Delta x$) is infinite. This means as soon as we increment our x-value, our error will jump to infinity, and we cannot recover from this. Secondly, all lines are drawn from left to right. If the first point on the line is to the right of the second point, the values are swapped.



Bresenham Algorithm from Wikipedia:

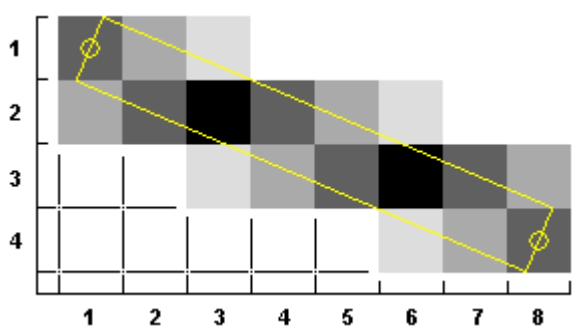
```
function line(x0, x1, y0, y1)
    boolean steep := abs(y1 - y0) > abs(x1 - x0)
    if steep then
        swap(x0, y0)
        swap(x1, y1)
    if x0 > x1 then
        swap(x0, x1)
        swap(y0, y1)
    int deltax := x1 - x0
    int deltay := abs(y1 - y0)
    real error := 0
    real deltaerr := deltay / deltax
    int ystep
    int y := y0
    if y0 < y1 then ystep := 1 else ystep := -1
    for x from x0 to x1
        if steep then plot(y,x) else plot(x,y)
        error := error + deltaerr
```

```
if error ≥ 0.5 then
    y := y + ystep
    error := error - 1.0
```

Line Rasterization with Antialiasing using Gupta-Sproull Algorithm

For the case when antialiasing is enabled, the lineRast module uses a variation of the Gupta-Sproull algorithm to rasterize antialiased lines. The algorithm draws lines that are three pixels wide and defines the color of each pixel depending on how far away the pixel is from the true line. The original algorithm relies on table lookups to convert from the distance to “coverage” however, we attempted to simplify the process by making pixel coverage a function of the distance from the true line.

In order to alter each pixels coverage, we remove the alpha values of the color provided and set it according to the pixel’s distance from the true line. In the case where the distance (e) is 0, the alpha value is set to its highest value, 255 (complete opaque). The larger distance gets, the smaller the alpha value becomes, making those pixels which are further from the true line more transparent. The alpha values are set as follows: alpha = 255 - (e << 1)



Gupta-Sproull Algorithm Variant from http://courses.engr.illinois.edu:

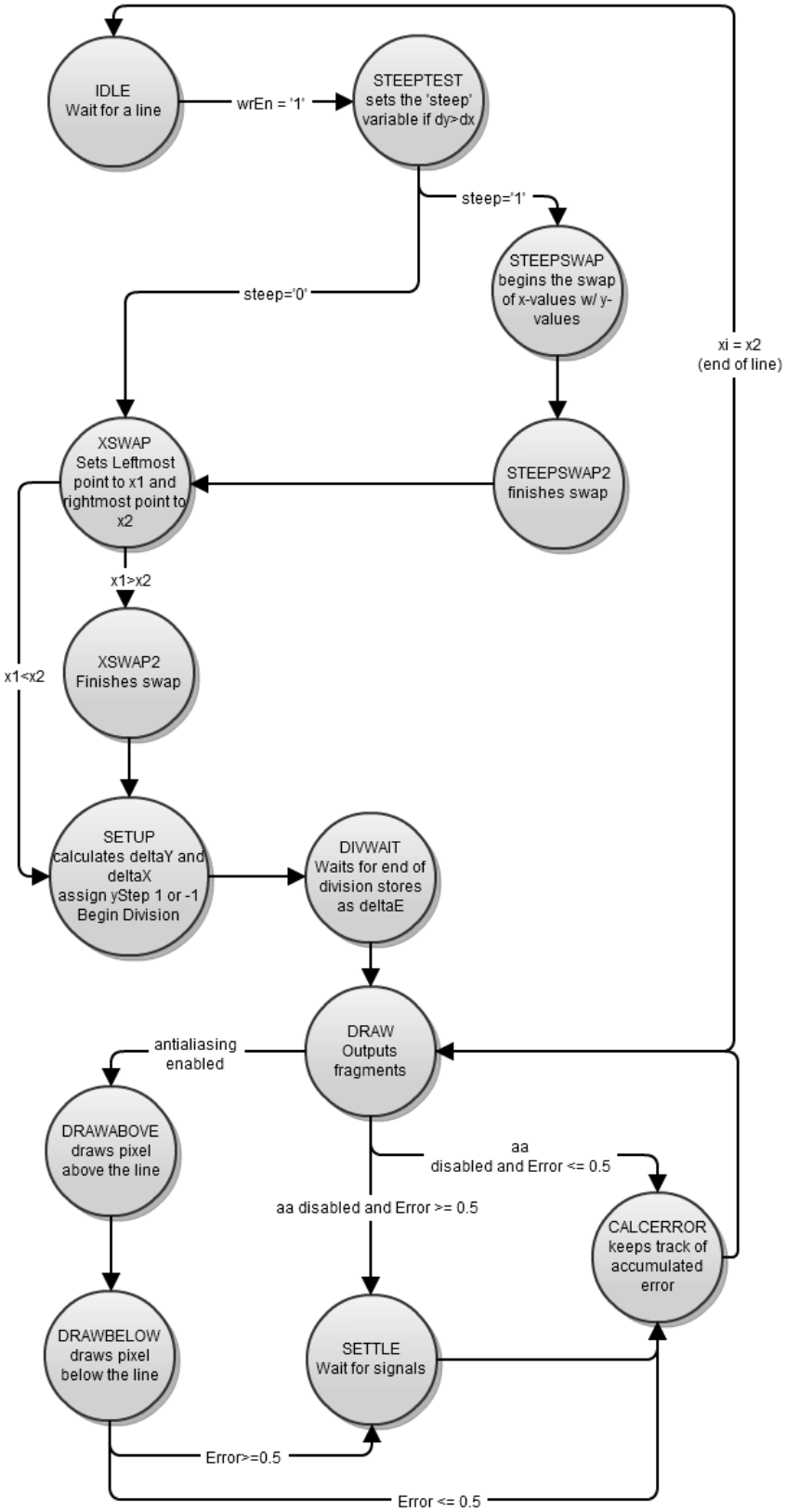
```
void AALine(int x0, int y0, int x1, int y1)
{
    int addr = (y0*640+x0)*4;
    int dx = x1-x0;
    int dy = y1-y0;
    /* By switching to (u,v), we combine all eight octants */
    if (abs(dx) > abs(dy))
    {
```

```
        /* Note: If this were actual C, these integers would be lost
         * at the closing brace.  That's not what I mean to do.  Do what
         * I mean. */
        int du = abs(dx);
        int dv = abs(dy);
        int u = x1;
        int v = y1;
        int uincr = 4;
        int vincr = 640*4;
        if (dx < 0) uincr = -uincr;
        if (dy < 0) vincr = -vincr;
}
else
{
        int du = abs(dy);
        int dv = abs(dx);
        int u = y1;
        int v = x1;
        int uincr = 640*4;
        int vincr = 4;
        if (dy < 0) uincr = -uincr;
        if (dx < 0) vincr = -vincr;
}
int uend = u + 2 * du;
int d = (2 * dv) - du;          /* Initial value as in Bresenham's */
int incrS = 2 * dv;          /* Δd for straight increments */
int incrD = 2 * (dv - du);  /* Δd for diagonal increments */
int twovdu = 0;  /* Numerator of distance; starts at 0 */
double invD = 1.0 / (2.0*sqrt(du*du + dv*dv));   /* Precomputed inverse denominator */
double invD2du = 2.0 * (du*invD);   /* Precomputed constant */
do
{
        /* Note: this pseudocode doesn't ensure that the address is
         * valid, or that it even represents a pixel on the same side of
         * the screen as the adjacent pixel */
        DrawPixelD(addr, twovdu*invD);
        DrawPixelD(addr + vincr, invD2du - twovdu*invD);
        DrawPixelD(addr - vincr, invD2du + twovdu*invD);

        if (d < 0)
        {
           /* choose straight (u direction) */
           twovdu = d + du;
           d = d + incrS;
        }
        else
        {
           /* choose diagonal (u+v direction) */
           twovdu = d - du;
           d = d + incrD;
           v = v+1;
           addr = addr + vincr;
        }
        u = u+1;
```

```
        addr = addr+uincr;
} while (u < uend);
```

Flow through LineRast



Resources

http://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm - Bresenham Algorithm
http://courses.engr.illinois.edu/ece390/archive/archive-f2000/mp/mp4/anti.html -Gupta Sproull Algorithm