Computer Graphics: Supervision 4

Daniel Chatfield

January 19, 2015

Past exam paper questions

1. 2012 Paper 4 Question 3

   (a) What are the main criteria to be considered in the design of a line drawing algorithm for a raster graphics display? [2]

   - Make the line appear to have uniform thickness.
   - Don't leave gaps in the line

   (b) Describe an algorithm to fill a series of pixels running from (x_0, y_0) to (x_1, y_1) that meets these criteria, explaining why it does so. Answers should consist of more than a fragment of pseudo-code. [6]

```
def draw_line(x0,y0,x1,y1):
    # The algorithm relies upon these assumptions:
    # - x0 < x1 (line goes left to right)
    # - the gradient is between -1 and 1 (otherwise the line
    #   is too steep and gaps appear)
    #
    # The following section of code checks these conditions
    # and swaps the coordinates around so that they are
    # satisfied

    flipped = False # True if x and y have been swapped with
                    # each other

    if abs(y1-y0) > abs(x1-x0):
        flipped = True
        x0, x1, y0, y1 = y0, y1, x0, x1

    if x0 > x1:
        x0, x1, y0, y1 = x1, x0, y1, y0

    delta_x = x1 - x0
    delta_y = abs(y1 - y0)
    error = delta_x / 2

    y = y0
    y_step = 1 if y0 < y1 else -1

    # For each column draw a single pixel, if error becomes
    # positive then we need to either increment or decrement y
    # for the next pixel.
    for x in range(x0, x1+1):
        if flipped:
            draw(y, x)
        else:
            draw(x, y)
        error -= delta_y
        if error < 0:
            y += y_step
            error += delta_x

    # This algorithm ensures that the line has uniform
    # thickness with no gaps by plotting a single pixel for
    # each column. For gradients not between -1 and 1 it swaps
    # them first (thus plotting a single pixel for each row
    # instead)
```

(c) A new volumetric display stores an image as a three-dimensional array of volume elements or voxels. Reformulate the design and implementation of the line-drawing algorithm to fill a series of voxels running from $(x0, y0, z0)$ to $(x1, y1, z1)$. [6]

```
def draw_line_3d(x0,y0,z0,x1,y1,z1):
    # The algorithm relies upon these assumptions:
    # - x0 < x1 (line goes left to right)
    # - delta_x > delta_y and delta_x > delta_z
    #
    # The following section of code checks these conditions
    # and swaps the coordinates around so that they are
    # satisfied

    x_flipped_with = False

    if abs(y1-y0) > abs(x1-x0) and abs(y1-y0) > abs(z1-z0):
        flipped = "y"
        x0, x1, y0, y1 = y0, y1, x0, x1

    if abs(z1-z0) > abs(x1-x0) and abs(z1-z0) > abs(y1-y0):
        flipped = "z"
        x0, x1, z0, z1 = z0, z1, x0, x1

    if x0 > x1:
        x0, y0, z0, x1, y1, z1 = x1, y1, z1, x0, y0, z0

    delta_x = x1 - x0
    delta_y = abs(y1 - y0)
    delta_z = abs(z1 - z0)
    error_y = error_z = delta_x / 2

    y = y0
    z = z0
    y_step = 1 if y0 < y1 else -1
    z_step = 1 if z0 < z1 else -1

    # For each column draw a single pixel, if error becomes
    # positive then we need to either increment or decrement y/z
    # for the next pixel.
    for x in range(x0, x1+1):
        if flipped == "y":
            draw(y, x, z)
        elif flipped == "z":
            draw(z, y, x)
        else:
            draw(x, y, z)
        error_y -= delta_y
        error_z -= delta_z

        if error_y < 0:
            y += y_step
            error_y += delta_x


        if error_z < 0:
            z += z_step
            error_z += delta_x
```

(d) How would this line-drawing algorithm be used to draw Bézier curves in three dimensions? [6]

1. Determine whether the line from p_0 to p_3 is an adequate aproximation by checking whether it is within a certain tolerance of the Bézier curve. If it is then draw the line and stop, otherwise continue.
2. Subdivide the curve into two bezier curves and return to step 1 for each.

2. 2011 Paper 4 Question 3 You are writing code for a new graphics card that is software programmable, rather than having the algorithms embedded in hardware. You want to write a fast triangle-drawing program to test the card.

   (a) Provide pseudocode, or similar, that draws a triangle with a constant colour. Assume that the inputs are the colour of the triangle and three two-dimensional points, representing the three vertices of the triangle. Further, assume that all three vertices lie on the visible screen and that no anti-aliasing is required. You may assume that there is a function to set a pixel, (x, y), to a particular colour, c, e.g. $setpixel(x, y, c)$, but you must provide pseudocode for any other functions that you require. Your answer should be sufficiently detailed that it could be transferred directly into a language such as Java but your answer does not, itself, have to be syntactically correct. [13]

```
def draw_triangle(x0, y0, x1, y1, x2, y2, c):
  if y2 < y1:
      x1, y1, x2, y2 = x2, y2, x1, y1

  if y1 < y0:
      x0, y0, x1, y1 = x1, y1, x0, y0

  if y2 < y1:
      x1, y1, x2, y2 = x2, y2, x1, y1
```

```
# coordinates are now sorted by y

gradient_1 = (x1 - x0) / (y1 - y0)
gradient_2 = (x2 - x0) / (y2 - y0)

for row in range(y0, y1):
    intersection_1 = x0 + (row - y0) * gradient_1
    intersection_2 = x0 + (row - y0) * gradient_2

    if intersection_2 < intersection_1:
        intersection_1, intersection_2 = (intersection_2,
            intersection_1)

    for col in range(intersection_1, intersection_2+1):
        setpixel(col, row, c)

gradient_1 = (x1 - x2) / (y1 - y2)

for row in range(y1, y2):
    intersection_1 = x0 + (row - y2) * gradient_1
    intersection_2 = x0 + (row - y0) * gradient_2

    if intersection_2 < intersection_1:
        intersection_1, intersection_2 = (intersection_2,
            intersection_1)

    for col in range(intersection_1, intersection_2+1):
        setpixel(col, row, c)
```

(b) Outline the extra steps required to draw a triangle specified by three-dimensional points in world space, where the triangle may extend beyond the edges of the screen after conversion to screen space. [4]

- Project the coordinates onto the 2D screen space.
- clip to the edge of the screen (splitting into smaller triangles)

(c) Outline the steps required to calculate the triangle's colour, assuming diffuse shading, with multiple point lights, but still producing a single colour for the whole triangle. [3]

For each pixel that is part of the triangle calculate the diffuse shading using the normal to the surface of the triangle and the direction to each of the light sources. Combine these to get the overall shading for that pixel.

Practical Programming

3. Implement one curve drawing algorithm in 2D. Note that your algorithm should use a line drawing algorithm to draw curve segments.

```
class Point():
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        if isinstance(other, Point):
            return Point(self.x + other.x, self.y + other.y)
        else:
            return Point(self.x + other, self.y + other)

    def __mul__(self, other):
        if isinstance(other, Point):
            return Point(self.x * other.x, self.y * other.y)
        else:
            return Point(self.x * other, self.y * other)

    def __div__(self, other):
        return Point(self.x / other, self.y / other)

    def flip(self):
        self.x, self.y = self.y, self.x


class Bezier():
    tolerance = 1

    def __init__(self, p0, p1, p2, p3):
        self.p0 = p0
        self.p1 = p1
        self.p2 = p2
        self.p3 = p3

    def distance(self, point):
        p = self.p3 - self.p0

        u = (point - self.p0) * p / float(p.x*p.x + p.y*p.y)

        if u > 1:
            u = 1
        elif u < 0:
            u = 0

        d = self.p0 + u * p

        return d.x*d.x + d.y*d.y

    def flat(self):
        distance = self.distance(self.p1) + self.distance(self.p2)

        return distance < self.tolerance

    def draw_curve(self):
        if self.flat():
            self.draw_line()
        else:
            left, right = self.subdivide()
            left.draw_curve()
            right.draw_curve()

    def draw_line(self):

        flipped = False

        if abs(self.p3.y-self.p0.y) > abs(self.p3.x-self.p0.x):
            flipped = True
            self.p0.flip()
```

```
                self.p3.flip()

        if self.p0.x > self.p3.x:
            self.p0, self.p3 = self.p3, self.p0

        delta_x = self.p3.x - self.p0.x
        delta_y = abs(self.p3.y - self.p0.y)
        error = delta_x / 2

        y = self.p0.y
        y_step = 1 if self.p0.y < self.p3.y else -1

        for x in range(self.p0.x, self.p3.x+1):
            if flipped:
                draw(y, x)
            else:
                draw(x, y)
                error -= delta_y
            if error < 0:
                y += y_step
                error += delta_x

    def subdivide(self):
        q0 = self.p0
        q1 = 0.5 * (self.p0 + self.p1)
        q2 = 0.25 * self.p0 + 0.5 * self.p1 + 0.25 * self.p2
        q3 = 0.125*(self.p0 + self.p3) + 0.375*(self.p1 + self.p2)
        r0 = q3
        r1 = 0.25 * self.p1 + 0.5 * self.p2 + 0.25 * self.p3
        r2 = 0.5 * self.p2 + 0.5 * self.p3
        r3 = self.p3
        return (Bezier(q0, q1, q2, q3), Bezier(r0, r1, r2, r3))
```