Computer Graphics SoSe16
Exercise 5

Tobias Knopp & Martin Hofmann
Institute for Biomedical Imaging

Next exercises:
20.06.2016, 14:15-15:45 & 16:30-18:00
27.06.2016, 14:45-15:45 & 16:30-18:00

The second approach to rendering is called rasterization. Here the idea is to loop over all the primitive objects to be rendered and find the pixels, which are influenced. To understand the basic concepts we will develop a small software rasterizer, which allows us to draw lines and triangles as primitive objects.

1. Aim of the rasterization stage is to draw geometric primitives. Drawing takes place in a so called frame buffer, which stores the pixel data to be displayed. Create yourself a `type FrameBuffer` containing the fields `nx` and `ny` to store the number of pixels in x- and y-direction as well as `data` to store the values of all pixel. The value of each pixel is expected to lie within the range 0-255. Choose the type of `data` accordingly.

   Write a method to initialize the `FrameBuffer` using two `Int` values for `nx` and `ny` only. Add a function to `clear!(buffer::FrameBuffer)` clear the `data` of the `FrameBuffer` (set the calue of each pixel to zero) and one `colorPixel!(buffer::FrameBuffer,x::Int,y::Int,color::UInt8)` function to set the color of a pixel to a specified value.

   Finally write a function to plot the content of the buffer using the `imshow` function from `PyPlot` (turn off interpolation), such that the value `0` is plotted black and the value `255` of your `FrameBuffer data` is plotted white (gray values in between).

   Running

```
s = FrameBuffer(51,51)
colorPixel!(s,25,25,0xff)
plot(s)
```

   should show a plot of a black square with a single white dot in the middle.

2. From this point on we will work within the coordinate system of the `FrameBuffer`. Within this Coordinate System the center of each pixel (i, j) is located exactly at the corresponding coordinates (i, j).

   As a first task you will rasterize a 2D-line segment starting at (x_0, y_0) and ending at (x_1, y_1). To do so You will need to color a series of pixels (set their value to `0xff`) such that all the pixels in the line are directly connected to their direct neighbour (located one up, down, left, right or in the diagonals). Try out the following methods, think about their limitations and catch all the cases with a warning (`warn("Case not covered")`) and escape the function before changing any pixel values.

For each algorithm draw the following line segments (if possible) into `Framebuffer(51,51)` and plot the result:

(a) $(47, 26)$ to $(5, 26)$

(b) $(5, 5)$ to $(47, 47)$

(c) $(5, 10)$ to $(47, 42)$

(d) $(5, 47)$ to $(47, 5)$

(e) $(24, 5)$ to $(28, 47)$

(f) $(26, 5)$ to $(26, 47)$

- The most simple Ansatz (`drawSimpleLine`) is to loop over all Integer values $x \in [round(x_0), round(x_1)]$ and calculate the corresponding y value by
$$y = m(x - x_0) + y_0, \tag{1}$$
with $m = (y_1 - y_0)/(x_1 - x_0)$. Rounding $y \mapsto round(y)$ then yields the pixel coordinate (x, y) to color.
- A simple extension to the last algorithm circumvents the problems related to slopes $m \in (\infty, 1) \cup (-1, -\infty)$. Simply change the loop to loop over all $y \in [round(y_0), round(y_1)]$ and evaluate
$$x = m^{-1}(y - y_0) + x_0 \tag{2}$$
instead.
- Still a lot of costly arithmetic operations have to be performed (multiplications in each iteration). Optimize the performance by rewriting the algorithm such that inside the loop only additions are done.

3. Even more sophisticated is to use the Bresenham's line algorithm for drawing. In its basic form this algorithm works for $x_0 >= x_1$ and slopes $m \in [0, 1)$ and uses the implicit form of equation (1)
$$F(x, y) = Ax + By + C = 0 \tag{3}$$
with $A = y_1 - y_0$, $B = -(x_1 - x_0)$ and $C = (x_1 - x_0)y_0 - (y_1 - y_0)x_0$. Here $x_0 \leq x_1$ and $y_0 \geq y_1$ are expected to be integer values, which can be achieved for any input by rounding.

   Starting at the initial position (x_0, y_0) the next candidate pixels to colorize are $(x_0 + 1, y_0)$ and $(x_0 + 1, y_0 + 1)$. The Bresenham's line algorithm colorizes the candidate, which is closer to the line. To answer which pixel is closer, one can evaluate equation (3) at the midpoint in between
$$F(x_0 + 1, y_0 + \frac{1}{2}). \tag{4}$$
   If the value of equation (4) is positive then the ideal line is below the midpoint and closer to the candidate point $(x_0 + 1, y_0 + 1)$. Otherwise, it is closer to the point $(x_0 + 1, y_0)$. All the

other points on the line are colorized similarly. Starting at position (x_i, y_i) evaluate equation (4) at positions $(x_i + 1, y_i + \frac{1}{2})$ and colorize one of the candidate pixels accordingly.

Write an implementation for this algorithm using `Integer` arithmetics only. If possible avoid any multiplications within the main loop. Draw the same line segments as in the last task.

4. Unfortunately, the Bresenham's line algorithm does only work for $x_0 >= x_1$ and slopes $m \in (-1, 0]$. Cover all other cases by a remapping of in and output variables (swapping x, y and changes of sign). Now you should be able to draw all line segments.

5. Similar to the drawing of lines there are many possible ways to draw triangles. One common method uses Barycentric coordinates. Consider a non-degenerate triangle defined by its three vertices $\vec{a} = (x_a, y_a)$, $\vec{b} = (x_b, y_b)$ and $\vec{c} = (x_c, y_c)$. Then each point $\vec{x} = (x, y)$ can be written as

$$\vec{x} = \alpha\vec{a} + \beta\vec{b} + \gamma\vec{c} \tag{5}$$

   with α, β, γ being its the unique Barycentric coordinates satisfying $\alpha + \beta + \gamma = 1$. They can be calculated by

$$\gamma = \frac{(y_a - y_b)x + (x_b - x_a)y + x_a y_b - x_b y_a}{(y_a - y_b)x_c + (x_b - x_a)y_c + x_a y_b - x_b y_a} \tag{6}$$

$$\beta = \frac{(y_a - y_c)x + (x_c - x_a)y + x_a y_c - x_c y_a}{(y_a - y_c)x_b + (x_c - x_a)y_b + x_a y_c - x_c y_a} \tag{7}$$

$$\alpha = 1 - \beta - \gamma \tag{8}$$

   For each point inside the triangle its Barycentric coordinates satisfy $0 < \alpha < 1$, $0 < \beta < 1$ and $0 < \gamma < 1$.

   Write a function `drawTriangle` which colorizes the pixels inside the triangle based on Barycentric coordinates. Draw and plot the triangle with coordinates $(10, 10)$, $(10, 42)$ and $(42, 42)$.

6. Another interesting point about Barycentric coordinates is, that they can be used to interpolate values like color across the surface of the triangle. Let k_a, k_b and k_c be the color assigned to each vertex of the triangle. For a point $\vec{x}$ with barycentric coordinates $0 < \alpha < 1$, $0 < \beta < 1$ and $0 < \gamma < 1$ the vertex colors can be interpolated by

$$k_x = \alpha k_a + \beta k_b + \gamma k_c. \tag{9}$$

   Write a function, which additionally takes vertex gray values as input arguments and draws the triangle with interpolated gray values for each pixel inside the triangle. Draw and plot the triangle with coordinates $(10, 10)$, $(10, 42)$ and $(42, 42)$ and the gray values `0x40`, `0x80` and `0xf0`.