Prof. Dr. F. Sadlo, D. Aumiller *Heidelberg, Dezember 05, 2016*

Exercise Sheet 7

Assignment 7.1 Rasterization II: Triangle Rasterization [11 Points]

After the triangles have been transformed to screen coordinates, they now have to be drawn to the screen. You can use your (correct) solution from the last exercise sheet. If you have not implemented all of the tasks from the last sheet, download the new skeleton from the elearning platform. The implementation will be done in the file `SimpleRasterizer.cpp` once again. If you implement additional functions in the file header, make sure to include the relevant files in your upload.

1. **Drawing of the Triangle**
   To completely display a triangle mesh, as loaded in the last exercise sheet, each individual triangle has to be drawn correctly. On the last sheet, the function `DrawTriangle()` only drew the vertices. This functionality should now be replaced with a scanline algorithm.

   For better comparison, it is advantageous if there is only one triangle being displayed, which should also not rotate. Therefore, set the variable `rotate` and `loadFile` in the `main` method (`main.cpp`) to `false`, or — alternatively — run your program with the command line parameters "`-norotate -nogeometry`". This will now only display a single triangle, of which you should only see the vertex pixels. You may comment out the existing code in the function `DrawTriangle()` for the rest of the task.

   The triangle t is already passed to `DrawTriangle()` with screen coordinates. Draw the triangle t with the scanline algorithm. Start by implementing the function `DrawSpan()`, which draws a horizontal line from x_1 to x_2 on the height of y. To display a point, use the function `image->setPixel()` in a way similar to the code in `DrawTriangle()`, given in the skeleton. For now, use `vec[1]` as a color value. Keep in mind that the value of x_1 does not necessarily have to be smaller than the value of x_2, and that `SetPixel()` does not perform any range checks. It is your responsibility to avoid drawing outside of the image. Draw the lines in such a fashion that the pixels will be included on the left side, but excluded on the right, to avoid overlaps within the triangle mesh. After that, you may complete the scanline algorithm in `DrawTriangle()` with the help of `DrawSpan()`. Respect the fact that, initially, the vertices of the triangle are not ordered. [*7.5 Points*]

2. **Gouraud Shading**
   Extend your algorithm with Gouraud shading, by performing barycentric/bilinear interpolation of the three vertex colors of the triangle t in its interior. The vertices of the test triangle are set to red, green, and blue, respectively. You can use the parameters `color1` and `color2` of the function `DrawSpan()` to set the color in `SetPixel()` accordingly. [*3.5 Points*]

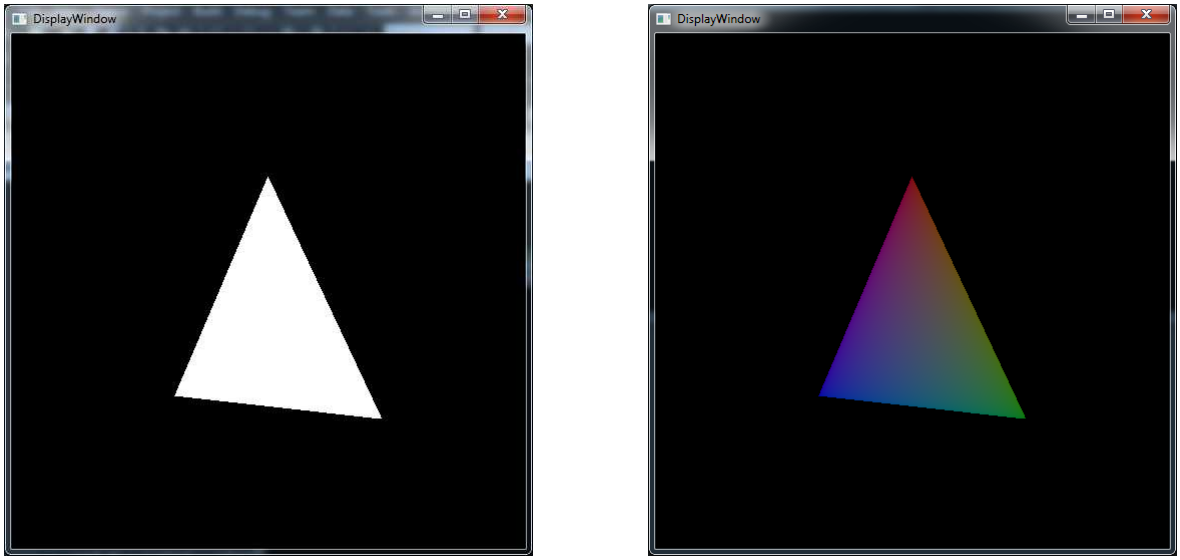

Figure 1: Filled triangles with white color (task 1.1), and with Gouraud shading (task 1.2).

Submission: Dezember 12, 2016, 6:00 pm via Moodle