Programming Lab 11

Triangle Rasterization Using a Z-buffer

The header file RasterUtilities.h from the previous assignment contains the declaration of a function which you did not implement in that assignment:

void FillTriangle(Raster& r, const Point& P, const Point& Q, const Point& R);

In this assignment, you are to implement this function, which should draw a solid triangle with vertices P, Q, R using the current color to the frame buffer associated to the raster instance r. The points P, Q, R are given in device coordinates, as described in the previous assignment. The Z-buffer is used to determine which portions of the triangle are visible; i.e., which pixels within the triangle are actually rendered.

For full credit, your code should meet the following criteria.

- Only pixels within (and on) the boundaries of the triangle should be rendered. Note that your code does not need to check if the points PQR lie within the frame buffer rectangle.

- Linear interpolation should be used to determine the z value for each pixel that is potentially rendered.

- Only those pixels that have a z value that is smaller than the current value in the Z-buffer at the corresponding pixel location should be rendered.

- The DDA method should be used when possible to avoid floating point multiplications. In particular, there should be no floating point multiplications inside of the innermost loop, and at most a single floating point multiplication in the outermost loop.

- Implicit integer multiplications should be avoided when possible by using IncrementX in place of GotoPoint. In particular, there should be no calls to GotoPoint inside of the innermost loop, and at most a single call to GotoPoint in the outermost loop.

Your submission for this part the assignment should consist of a single source file, named FillTriangle.cpp.

Remark. In debug mode (which is the default mode), the WritePixel, WriteZ, and GetZ member functions of the Raster class will cause an assertion failure whenever an attempt is made to access memory locations outside of the frame bu er. This is not always convenient when testing code. You can force per pixel clipping to the frame bu er by making the macro definition CLIP PIXELS. With the MSVC compiler this can be done on the command line using the /D switch:

```
cl /EHsc /DCLIP_PIXELS ...
```

when compiling your code.