Programming Lab 10

The header file RasterUtilities.h contains the following function declarations

void FillRect(Raster& r, int x, int y, int w, int h);
void FillRectZ(Raster& r, int x, int y, int w, int h, float z=1);
void DrawLine(Raster& r, const Point& P, const Point& Q);

(the header files Raster.h and Affine.h have been included). The Raster.h header file is similar to one that we used in DT284, but also includes additional functionality for a Z-buffer, namely the WriteZ and GetZ functions for setting and retrieving values in buffer.

The Raster class is detailed at the end of this handout.

FillRect(r, x, h, w, h) – sets the pixel values of the frame buffer (associated o r) to the current color. Only values within the rectangle with lower left hand corner (x, y) and with width w and height h are affected. That is, values in the frame buffer at pixel coordinates (i j), where

$$x \leq i < x + w \text{ and } y \leq j < y + h$$

are set to the current color.

FillRectZ(r, x, y, w, h, z) – sets the values of Z-buffer to z (default value is z = 1, the far plane). Only values within the rectangle with lower left hand corner (x, y) and with width w and height h are affected.

DrawLine(r, P, Q) – draws a line segment between the points P and Q, both given in device coordinates, to the frame buffer. The Z-buffer is used to determine which portions of the line segment are visible (and thus are rendered). The device coordinate system maps the camera view frustum to the axis aligned rectangle.

$$0 \leq x \leq W - 1, 0 \leq y \leq H - 1, -1 \leq z \leq 1$$

where W, H give the width and height of the frame buffer, the plane z = 1 corresponds to the near plane of the view frustum, and z = 1 corresponds to the far plane.

Note that part of your grade will be for the effciency of your coding of the above functions. In particular, you should use the DDA idea in the DrawLine function.

Your submission for this part the assignment should consist of a single source file, named RasterUtilities.cpp. You may include only the header files RasterUtilities.h (which includes Raster.h and Affine.h) and the standard header file cmath.

The Raster Class

The file Raster.h contains the following declarations.

```
typedef unsigned char byte;
Raster(byte *rgb_data, float *zbuffer , int W, int H, int S);
void GotoPoint(int x, int y);
void SetColor(byte r, byte g, byte b);
void WritePixel(void);
void WriteZ(float z);
float GetZ(void);
void IncrementX(void);
void DecrementX(void);
void IncrementY(void);
void DecrementY(void);
```

Raster(rgb_data, zbuffer, W, H, S) – (constructor) creates an instance of the Raster class associated to the frame buffer starting at address rgb_data, and with Z-buffer starting at address zbuffer. The frame buffer and Z-buffer are both W pixels wide, and H pixels tall. In the case of the frame buffer, each scan line takes up S bytes (the stride of each scan line). Each scan line of the Z-buffer takes up W sizeof(float) bytes (the stride value S only applies to the frame buffer). The caller of constructor is responsible for allocating the memory for the frame buffer and Z-buffer, and the memory is expected to persist over the lifetime of the class instance.

GotoPoint(x,y) – sets the current point to pixel location (xy). No data is written to the frame buffer or Z-buffer.
SetColor() – sets the current foreground color.
WritePixel() – writes a pixel to the frame buffer at the current point in the current foreground color. See the comments below.
WriteZ(z) – writes the value z to the Z-buffer at the current point.
GetZ() – returns the value in the Z-buffer at the current point.
IncrementX() – moves the current point one pixel to the right.
DecrementX() – moves the current point one pixel to the left.
IncrementY() – moves the current point one pixel upwards.
DecrementY() – moves the current point one pixel downwards.

By default, the function WritePixel will cause the program to abort if the current point lies outside of the screen rectangle. Specifically, if (xy) is the current point, then when WritePixel is called, then four comparisons

$$0 \le x \text{ and } x < W \text{ and } 0 \le y \text{ and } y < H$$

are performed. If any of the comparisons are found to be false, the program will be aborted.

This behavior can be modified in two ways. First, if the symbol NDEBUG is defined, then none of the above comparisons are performed, and the pixel will be written. If a point outside of the screen rectangle is written to, undefined behavior will result. Second, if the symbol CLIP_PIXEL is defined, the above comparisons will be performed. If any of the comparisons are found to be false, the pixel will not be written to the frame buffer.

Remark on efficiency. The functions IncrementX, DecrementX, IncrementY, DecrementY are more efficient than the function GotoPoint. Indeed, the former functions only require a single addition to update the current point. In contrast, the latter requires a multiplication as well as an addition to compute the current point. For this reason, the GotoPoint should be avoided in favor of the incremental functions whenever possible (however, this is not always possible).