# Programming Assignment #6

CS 200, Fall 2015

*Due Thursday, October 29*

Implement the DDA triangle scan conversion algorithm discussed in class. Recall that the header file `RasterUtilities.h` contains the prototype for such a function:

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

The function `FillTriangle` should scan convert the solid triangle with vertices $P, Q, R$ to the frame buffer associated with the passed–in `Raster` class instance.
For full credit, your code should meet the following criteria.

- Only the header file `RasterUtilities.h` may be included (note that the the `Raster.h` and `Affine.h` headers file are included as a consequence). This means that you will have to write your floor and ceiling functions.
- Only pixels within (and on) the boundaries of the triangle should be rendered. You do not need to check if the points $P, Q, R$ lies within the frame buffer rectangle.
- You may not use any explicit multiplications *inside* of any loop (outside of a loop is okay).
- 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 in the outermost loop.
- Pixels should not be duplicated.

Your code will be judged on efficiency as well as cleanliness and clarity.

Your submission for this assignment should consist of a single source file, which should be named `FillTriangle.cpp`.

**Remark.** In debug mode (which is the default mode), the `WritePixel` member function of the `Raster` class will cause an assertion failure whenever an attempt is made to access memory locations outside of the frame buffer. This is not always convenient when testing code. You can force per–pixel clipping to the frame buffer 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.

**Suggestion.** Develop your code first without implementing the DDA improvements. Once you have the code working correctly, then you can try your hand at making your code more efficient by implementing DDA to avoid floating point multiplications.