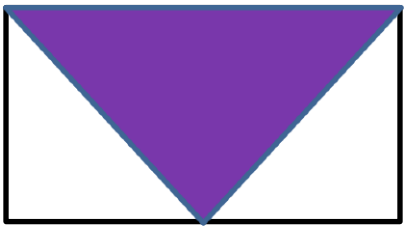
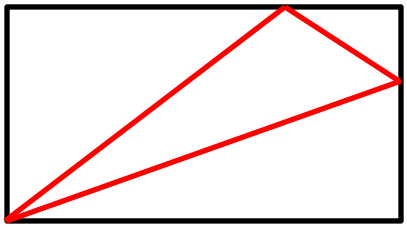
It's kind of a long story to explain everything clear.

1. The trityp4 file, performance is not improved, and about 1.5 times slower, the reason is that there are only 4 triangles, simply traverse doesn't have too much drawback. Theoretically, the hash table method should be faster, however, that depends on the method to generate all the points in an image, more specifically, what I used is the algorithm to generate every point inside a triangle. The algorithm I used now is to first getting a rectangle bounds which just includes the triangle and then traverse every point in this rectangle to test whether it's inside the triangle, this method is not the most efficient one, suppose there are n points inside a triangle, my method needs to test at least 2n points. The following graph shows the case for trityp4, there are 4 triangles, supposing each triangle contains n points, the hash table method needs to test at least 8n times, averagely the arraylist method also needs to test 8n times, ideally, those two methods should perform the same, but there are some extra tests need to do in the hash table method, so the hash table method is slower.



The above graph shows the best case, at some other cases, more time is wasted to generate the points inside a triangle, as showed below. When I want to generate all the n points inside the red triangle, the current hash table methods waste more time than n.

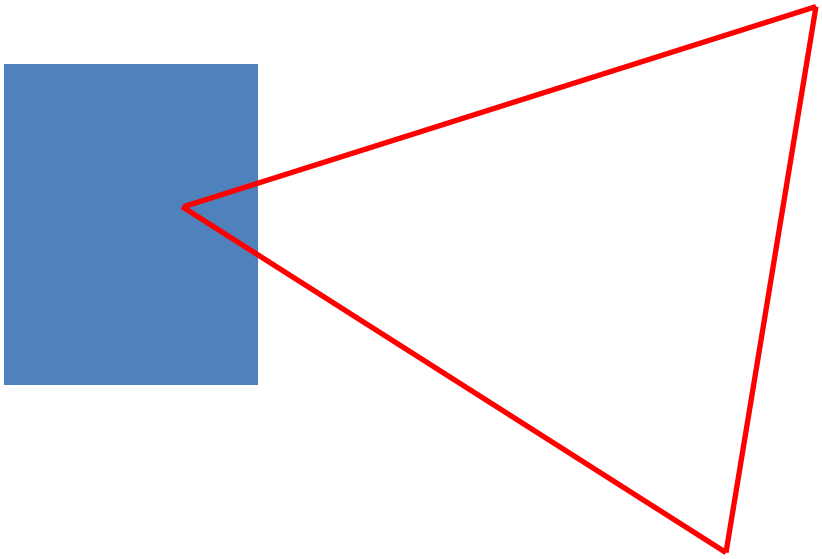


But why this method gives a large improvement in the type 6 and 7 shading, the reason is that in order to generate a smooth edge of each patch, at least 8 triangles are created for each patch, and each instance contains always more than one patch, a simple arraylist traverse wastes more time comparing to the generation method. The hash table method also shows advantage when the shading image is just a small portion of the whole effective drawing box.

The way to improve the current hash table method is to think about a more efficient algorithm to generate all the points in a triangle. I thought about some, but not as simple as this one, and a change in this method will require a thorough test in many files, before doing, it's hard to tell the side effect it will create.

2. The pattern-shading-2-4-idMatrix file, last time's code seems running endless, this is because the current hash table method needs to generate every points inside a triangle, when the triangle's

vertices are out of the device screen and far far larger than a reason value, like 50 times large, it will cause an almost endless traverse test. A schematic graph is showed below, the blue background is device screen, and the red triangle only has a small part inside the screen. The arraylist method only test every points inside the screen, so this case won't create a problem, but for the hash table method, it needs to generate every points inside the triangle according to the triangle's vertices' coordinate, it will create a problem in the code of last time. After I found this reason, I passed the deviceBounds information to the shading context instance, this issue can be avoided. At most time, there won't be such a problem, although the decode value can give some constrains, theoretically, it won't always ensure the mapped coordinates within the device screen.



The hash table method and the arraylist method almost consumes the same time for pattern-shading-2-4-idMatrix file.

3. The eci file, I check it again, the original type 4 and 5 shading are not that correct, the current code looks working fine , type 4 image is rendered right, type 5 image looks not right, but I doubt that is some issue in the color space, not the reading and calculation.
4. As I know the parameter deviceBounds can restrain an image, I can modify the hash table in type 6 & 7 shading to a 2D array, however, didn't do that yet and wait for your opinion.
5. Yes, there is no amazing result for this implement, however, I prefer the hash table method, once the algorithm of generating points inside a triangle is good enough, they will work fine, arraylist has potential problem when the size of the triangle list is large. But this is just my opinion, I respect your decision. I also know that there is other drawback in my code, when rendering a high resolution image, it still creates some unexpected spots for some files, this is also an issue in the CoonsTriangle class, however, to fix this and to improve the previous generation method are not in my plan for this project, even you require, I still cannot accomplish it, the current result is far more better than my expectation and the requirements of the project at the beginning. But if you or someone has a really good idea to improve the code, I will be happy to try.
6. I will continue working on this project either according to your suggestion or according to my own understanding of the potential problems needed to fix for this shading package. To test the

type 5 shading, I will try to find some type 5 shading test files; however, I will be more than happy to get some more type 5 shading test files from you.