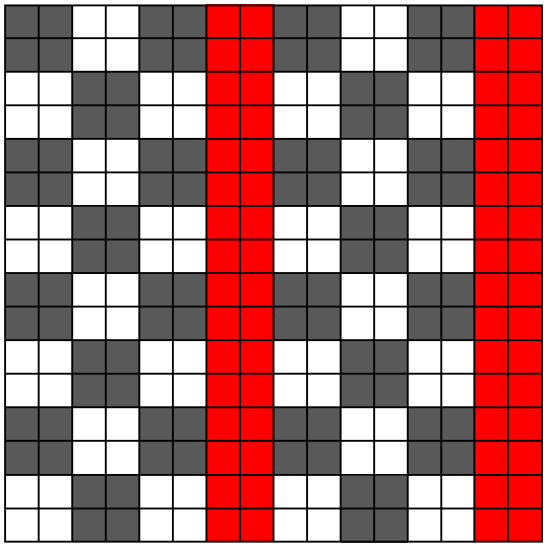
Stanford CS248: Interactive Computer Graphics
Exercise 2

Sampling, Prefiltering, and Texturing

Consider a 1024×1024 checkerboard texture where each check is a 2×2 square of similarly colored pixels, and every 4th check in the horizontal direction is red, as shown below. (not all 1024×1024 pixels are shown)



A. (3 pts) In the diagrams below draw levels 1, 2 and 3 of the mipmap resulting from this texture. Note that mipmap level 0 is the original texture. You can use 'RG' to denote reddish gray, and 'G' to denote gray if you wish.

B	W	B	R	B	W	B	R
W	B	W	R	W	B	W	R

G	RG	G	RG
G	RG	G	RG

RG	RG
RG	RG

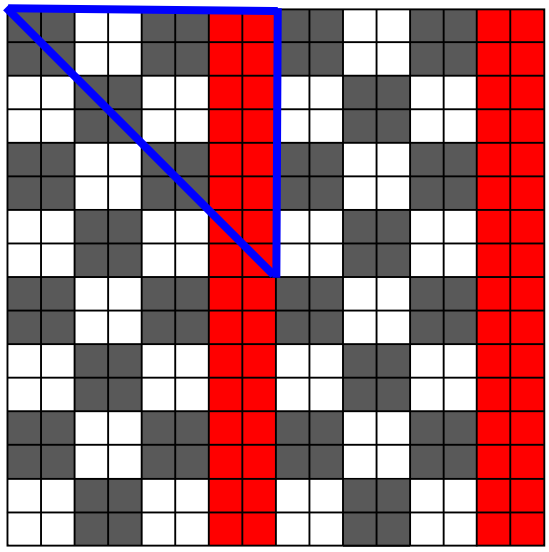
B. Consider a triangle with vertex screen positions (0,0), (0.5,0), and (0.5,0.5) rendered onto a 1024×1024 screen. The triangle has texture coordinates (0,0), (1,0), and (1,1) corresponding to these vertices. Assuming that we render the triangle without mipmapping (so, only accessing level 0) and bilinear filtering, what does the image look like, and why (describe the spacing of the texture samples in screen space)?

since a larger texture area maps to a smaller screen area, we have minification. Aliasing occurs since the texture samples are more than pixels and the spacing is narrow.

C. Now consider the case where triangle vertices are given screen positions (0,0), (1/8,0), (1/8,1/8). Again, like in A, no mipmapping is used, but bilinear filtering is used. Describe the appearance of the triangle now (describe the pattern you see, and what colors are seen). **In your answer comment on whether aliasing is occurring and why?**

There should be greater aliasing such as jaggies and Moire patterns. The texture samples are even greater than the nubmer of pixels than before.

D. We've copied an image of the original texture below for convenience. Assuming the triangle vertex positions are the same as in part C above, draw the region of texture space corresponding to the part of the surface that is visible in pixel (0,0) on screen. (Just draw a region on the figure.)



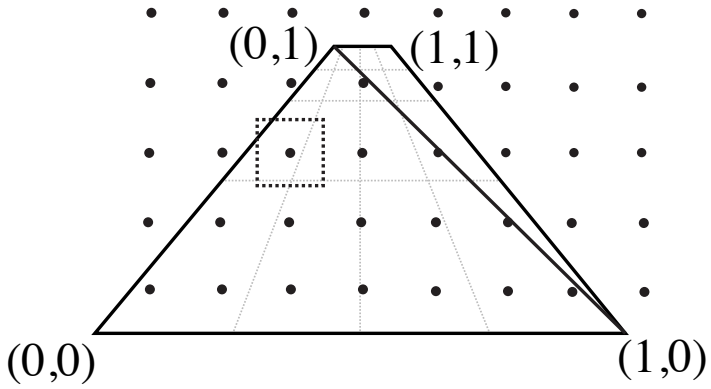
E. Imagine that the fragment shader, when processing the fragment for pixel (0,0) for the triangle in parts C and D, calls the GLSL texture function `sample2D(myTexture, uv)`. Assuming that **THERE IS STILL NO MIPMAPPING ENABLED** describe an implementation of `sample2D` that avoids artifacts due to undersampling. There is only one call to `sample2D` per fragment, and there is exactly one fragment per pixel.

We can use super-sampling instead of mipmapping. Average the values of the texels enclosed by the fragment, and use that average for the pixel value.

F. Now imagine the conditions are the same as in part E, except now mipmapping is enabled in the implementation of `texture2D`. Describe how mipmapping allows for exactly the same solution as in the previous problem, but at significantly lower cost.

it uses a precomputed and lower resolution texture map to avoid undersampling, instead of having to average the values of texels for each fragment during runtime.

G. Now consider using the texture map on a ground plane, the camera looking toward the horizon. Consider shading the highlighted pixel. Describe why in this case, a single mapmap lookup cannot be used to determined the properly prefiltered answer.



This is because the pixel's texture footprint does not map to an isotropic area in texture space, ie. the vertical resolution greater than the horizontal resolution. If a higher mipmap level is used, there would not be undersampling vertically, but there is insufficient resolution in the horizontal direction (downsampled), which results in blurring. If using a lower level, there would be enough texels per pixel horizontally, but there may cause aliasing vertically. Using more than one mipmap level can mitigate this.

H. (THIS PROBLEM IS OPTIONAL) Consider a system WITHOUT MIPMAPPING, but with a rasterizer that is configured to supersample coverage N times per pixel (N samples per pixel). Each of these samples generates a fragment, and the fragment is shaded using a `texture2D` lookup as before. ASSUME THAT THE IMPLEMENTATION OF `texture2D` only performs a single bilinearly interpolated texture lookup per call. Describe why this scheme, as N increases, will also accurately approximate the result given in Part E.