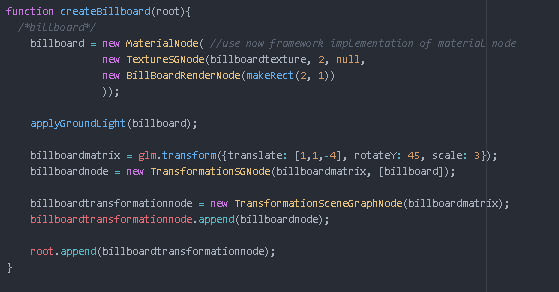
**BILLBOARD:**



Setup for the billboard. The important thing for billboarding on this screenhot is billboardrendernode where we enable a boolean in our shader.



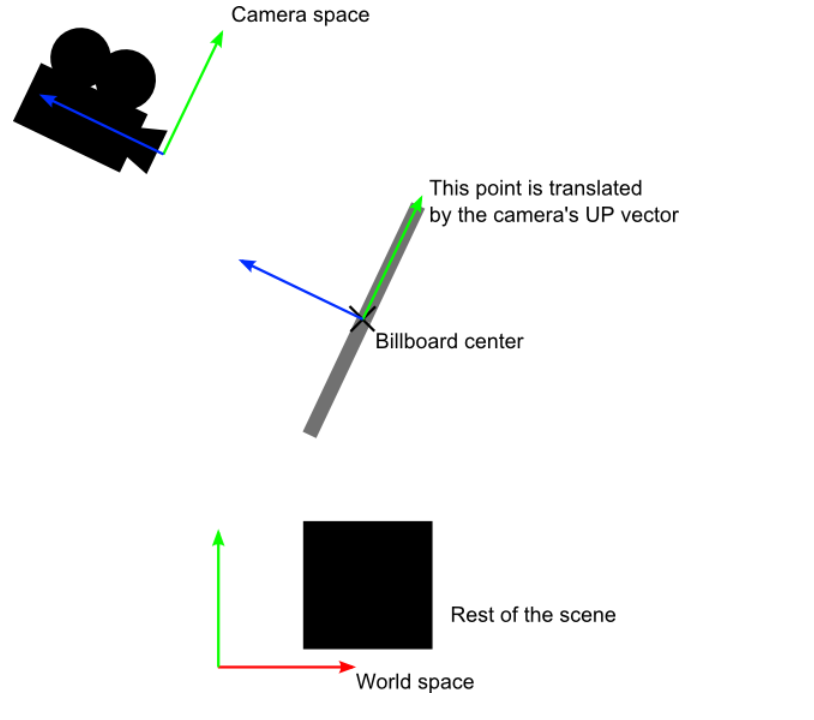
This Class ist there for enabling a boolean in a shader to apply billboarding.



***THEORY:***

Each corner of the billboard is at the center position, displaced by the camera’s up and right vectors.

We only know the billboard’s center position in world space, so we also need the camera’s up/right vectors in world space.



<http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/billboards/>

**Our implementation.**

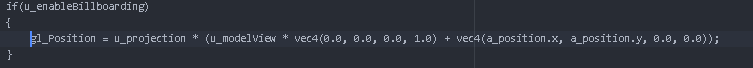
The magic happens here:

Normally gl\_position is computed by: vec4 projection matrix \*vec4 eyePosition matrix,

eyeposition is computed by: vec4 u\_modelView \* vec4(a\_position,1)

to achieve billboarding we take the projection matrix and multiply it with …

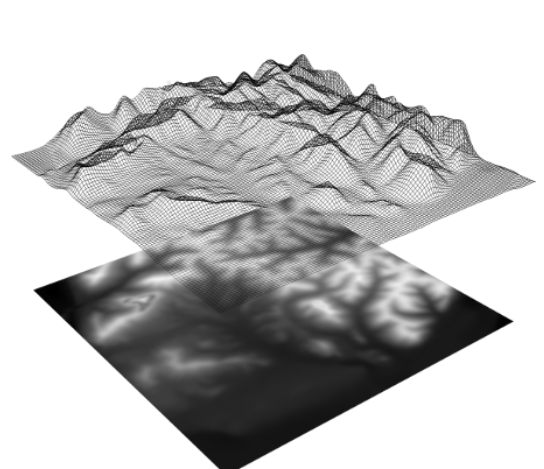
* Our modelView \* vec4(0,0,0,1) + the position.x and position.y as an offset.



**HEIGHTMAP:**

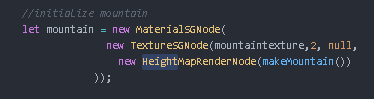
***THEORY:***

Heightmap is a gray scale image that is used to vary the height of a surface. Given a flat plane, the height of a surface that “floats” above the plane is specified by the colors in the gray scale image. The image values, which are intensities in the range [0.0, 1.0] , are considered percentages of a maximum height.

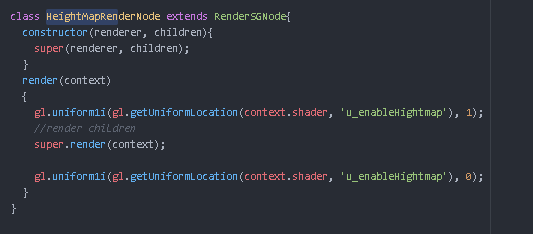


https://runestone.academy/runestone/static/learnwebgl2/11\_surface\_properties/09\_heightmaps.html

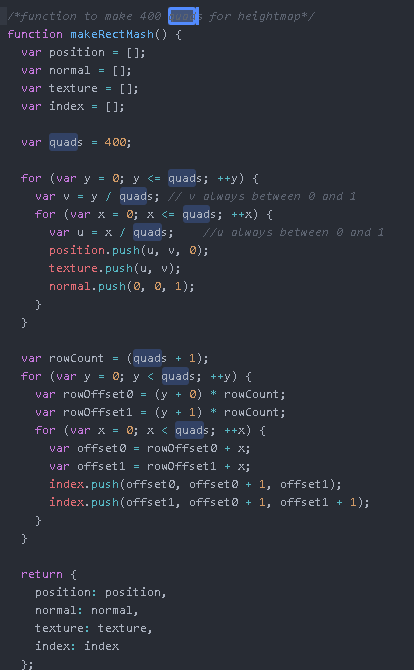
Setup for heightmap:



This Class ist there for enabling a boolean in a shader to apply heightmapping



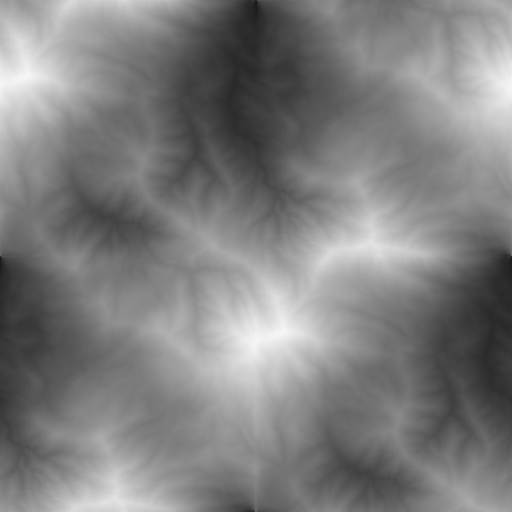
We need a quad that consists of a lot of vertices. Therefore we used this utility function for generating this quad. The more vertices the more granulare our heights in the surface are computed.

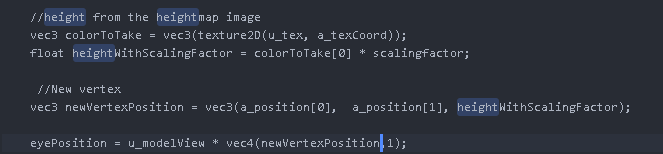


The height of the surface that lifts our flat plane is generated by the colors in this picture

We have a grey scale image, so all pixel colors have an equal amount of red, green and blue.

So we simple can use any of these three value for our heights.





The magic happens here in the shader

We fetch the color oft he heightmap at a vertex location using ist corresponding texture coordinates.

* Float colorToTake= vec3(texture2D(u\_tex, a\_texCoord)

We calculate the heght of this vertex by ist color value and scale it.

* Float heightWithScalingFactor= colorToTake [0] \* scalingfactor

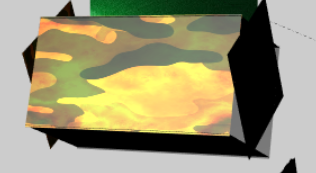
(= colorToTake [0] represents the red color [we could chooe colorToTake [1] or colorToTake [2] as well)

We use the height as the z component oft he vertex and creating a the new vertex

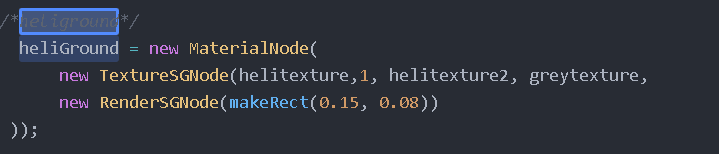
* Vec3 newVertexPosition = vec3(a\_position[0], a\_poition[1], heightWithScalingFactor)

**MULTITEXTURING**

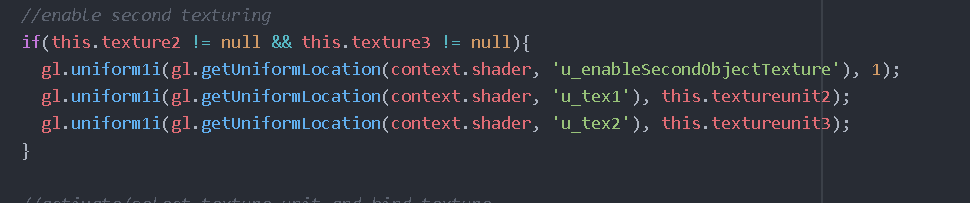
We applied multiptexturing to our manually composed modelt hat consits of multiple party (helicopter)



We pass three textures to our TextureSGNode



When passing a second and third texture we are enabling the second object texutre boolean fort the shader



We fetch the texturecolor oft he second texture by the texture and cooresponding texture coordinates

We want to blend the three textures equaly so we take the texturecolor(1)(2)(3).x add them together and divide them with 3 to have our new computed final texturecolor.x. We apply the same with y and z.

