A1-REPORT

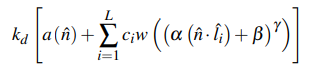
Evyn Brouwer – 100702629

Anthony Smiderle – 100695532

Daniel Hong – 100623669

MATH & ALGORITHM

TF2 Lighting (Unimplemented):



kd = albedo of the object sampled from the texture map

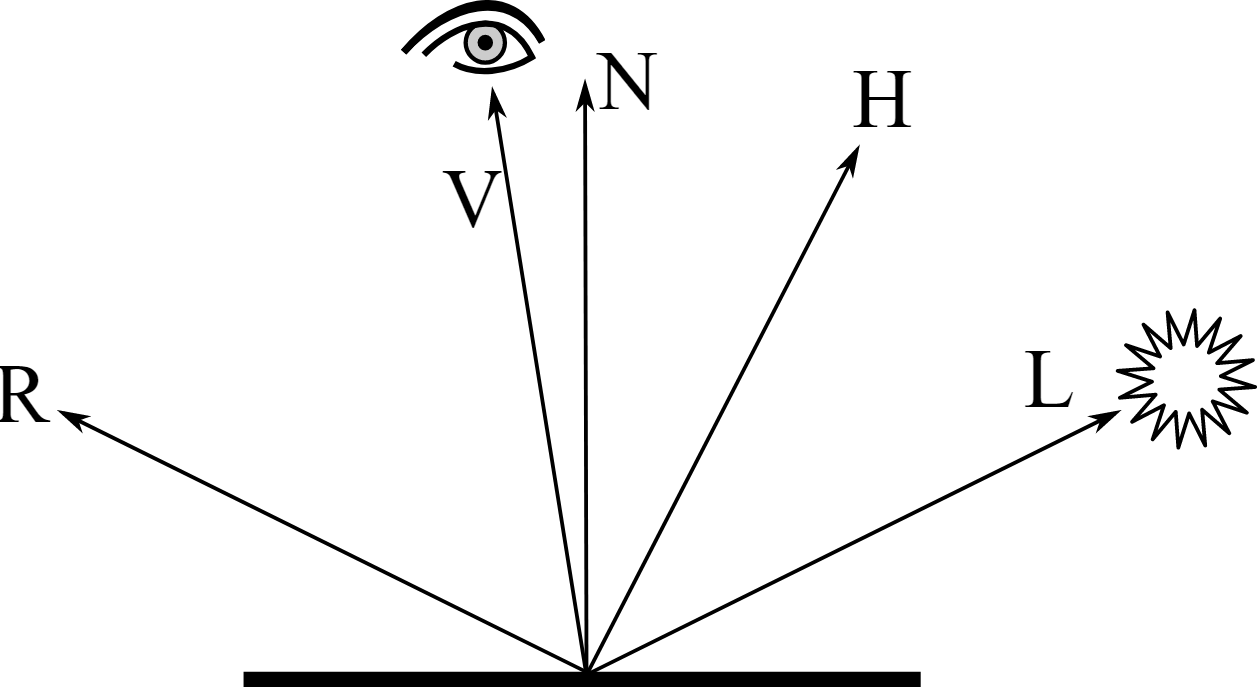
a(n) = directional ambient lighting on the object as a function of the per pixel normal n

∑Li=1 = the summation of the following function for L = number of lights and i = light index

ci = color of the light i

w = wrapping function that applies the 1D LUT below and multiplies the color output by 2 before clamping to 0-1 range

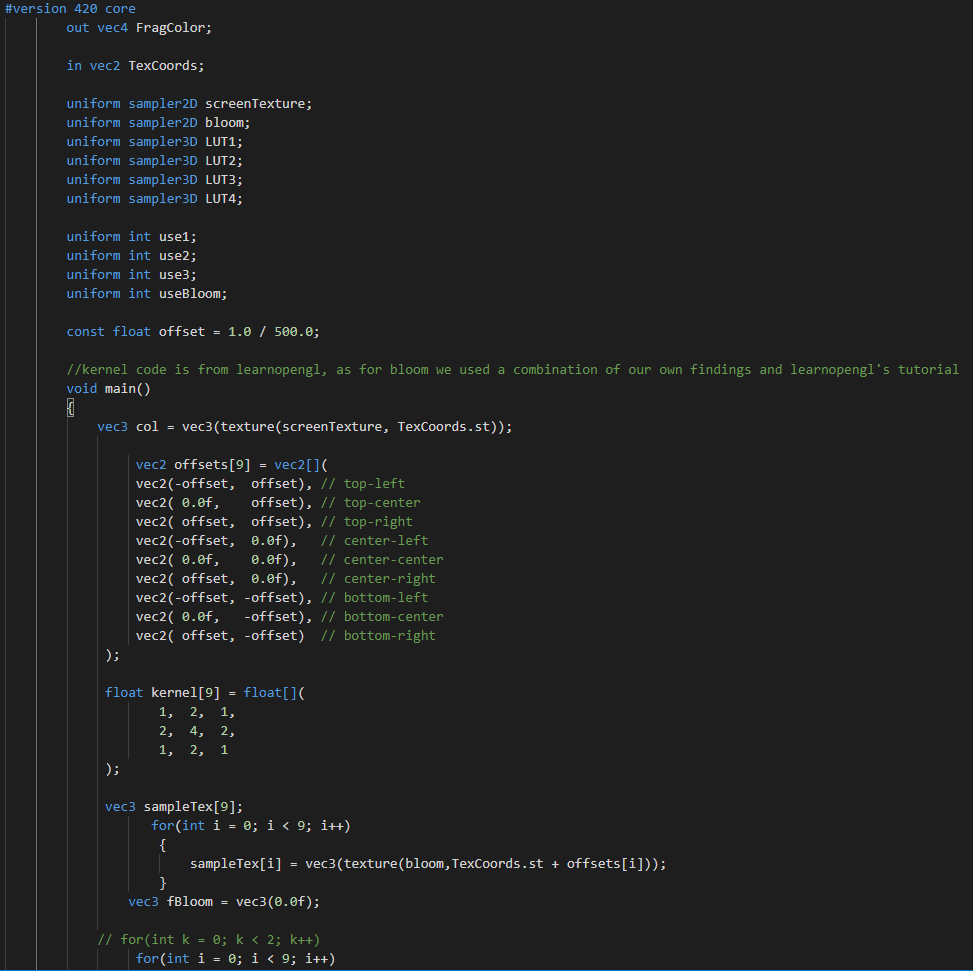
n dot li = Lambert term found through (N dot L) based on the following diagram:



a, b, y = scale, bias, and exponentiation which are set to 0.5, 0.5, 1 and for the purpose of TF2 lighting

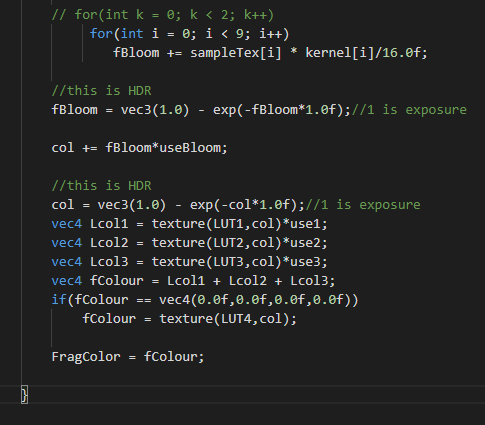
CODE

Post-Processing Frag Shader Pt. 1



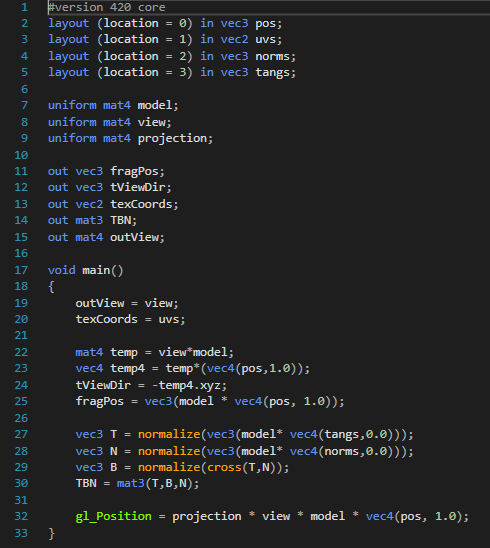
Theres a lot to unpack here, so lets start with framebuffers. A framebuffer with 2 colour buffers is used to have the regular screen texture and the bloom texture. Some offsets are calculated, and after sampling is complete a blur kernel is applied to the bloom texture (see below)

Pt. 2



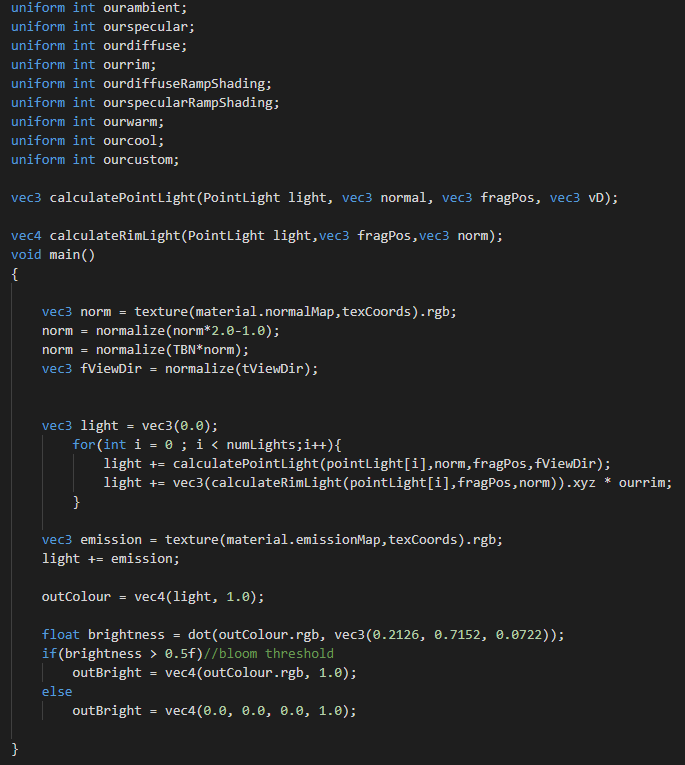
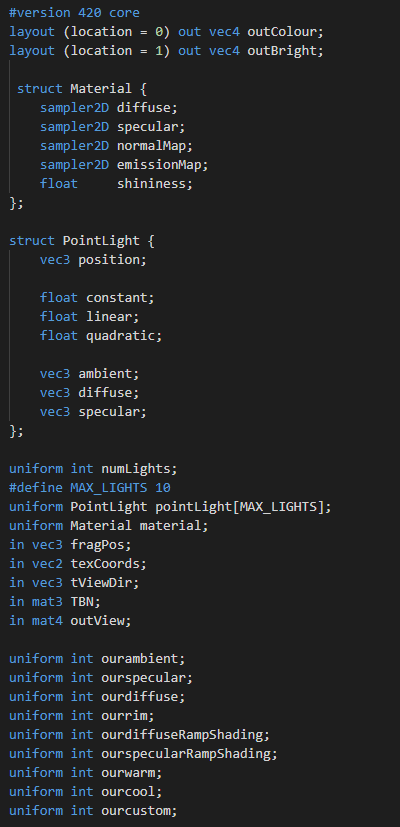
After bloom is applied, HDR is applied to the bloom itself. Bloom is then added to the final colour if the “useBloom” bool is true. HDR is added to screen colour using exposure mapping. The LUTs are all combined, and work the same as the bloom colour with their respective bools. If the final colour is black, then we should use the blank LUT in combination with the screen colour.

Vertex



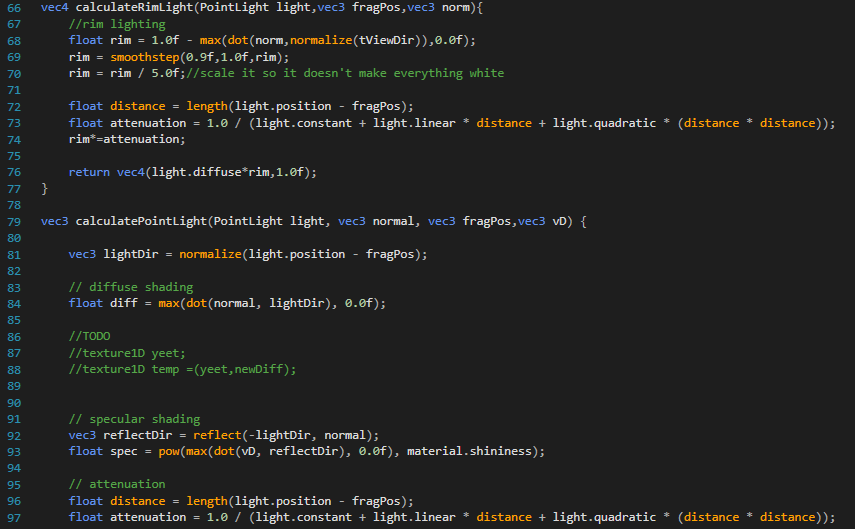
This fragment shader sends the texture coordinates, view direction, frag position and TBN matrix to the fragment shader. Besides that, it combines model view and projection matrices with the input position to project the model into 3D space. The TBN matrix is used in order to achieve tangent-space normal mapping.

Fragment

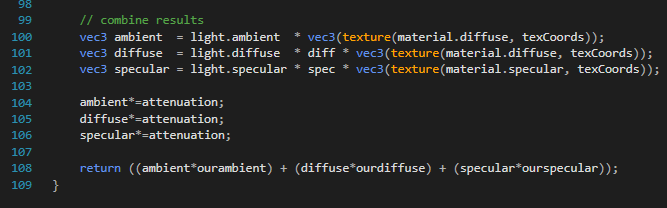


This first screenshot just shows how we organize the data from the vert shader/C++ side of things in the frag shader. The 2 layout locations are for rendering to the screen texture / bloom texture respectively.

The second screenshot shows in a nutshell how we calculate light, along with how bloom is done.



Here we calculate rim lighting with some custom scaling parameters and attenuation, cause we thought it would look cool. Along with that is Phong shading, particularly a point light. reflection direction is calculated, then attenuation, and all the results are combined together with the light’s specific variables and the textures passed. Attenuation is combined with all the components, and then all components are multiplied by their bools in order to have our toggle effects.



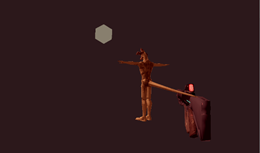
Screenshots of Modes

No lighting Ambient Lighting

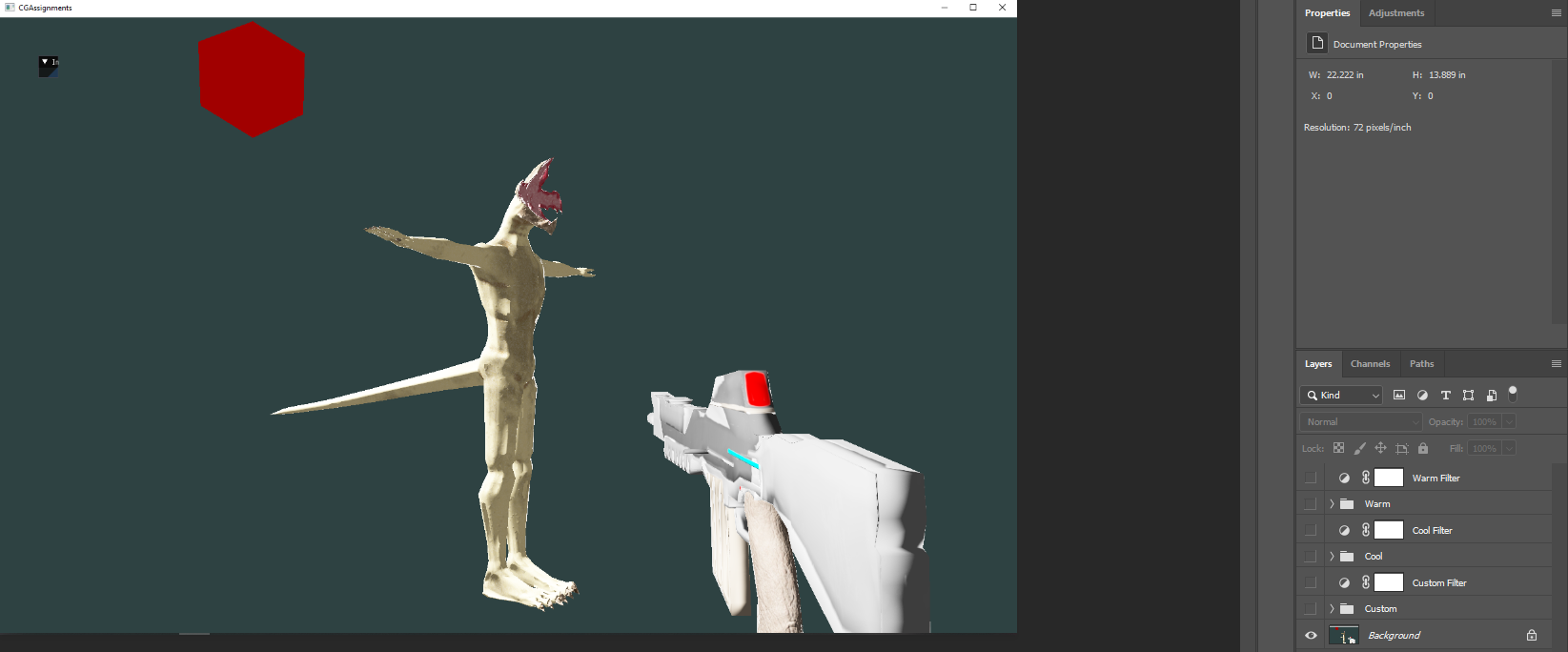
Specular Lighting Specular and Rim Lighting

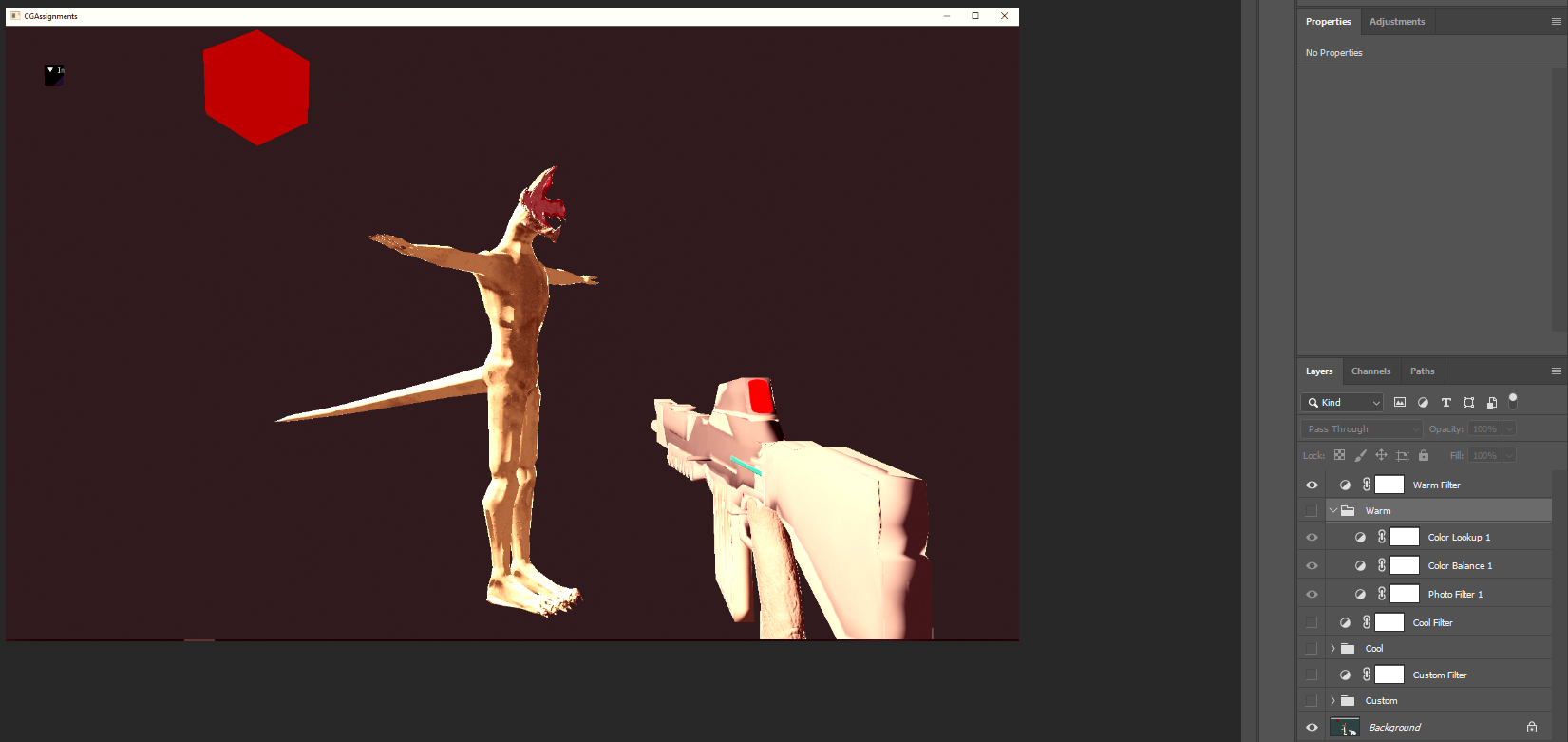
Specular, Rim, and Ambient Lighting Warm Colour Grading

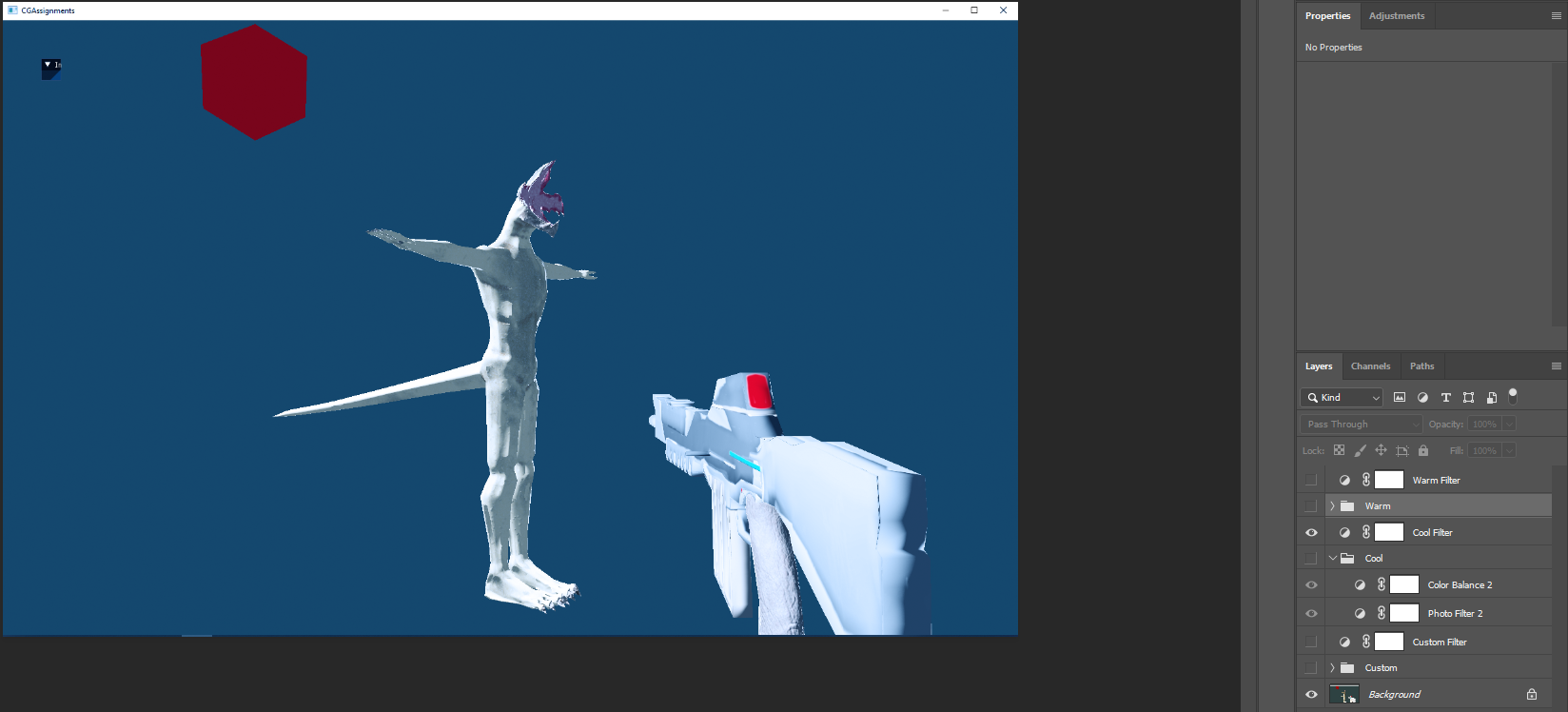
Cool Colour Grading Custom Colour Grading



Base



Warm LUT



Cool LUT



Custom LUT

References:

<https://developer.nvidia.com/gpugems/gpugems2/part-iii-high-quality-rendering/chapter-24-using-lookup-tables-accelerate-color>

<https://wwwimages2.adobe.com/content/dam/acom/en/products/speedgrade/cc/pdfs/cube-lut-specification-1.0.pdf>

<https://steamcdn-a.akamaihd.net/apps/valve/2007/NPAR07_IllustrativeRenderingInTeamFortress2.pdf>

<http://in2gpu.com/2014/07/02/rim-lighting/>