3D Graphics Programming

Andrew Richardson - W9383619

Assessment Report

Andrew Richardson W9383619
10/01/2022

Table of Contents

Andrew Richardson W9383619
10/01/2022

Introduction

Scene Features

Implemented as specified in the ICA document.

- **Creation and rendering of a cube**
  - The generation of vertices and triangles to render a 3D cube
  - Each side is rendered in a different colour
  - The cube rotates
- **Rendering of a gridded mesh to represent terrain**
  - The correct implementation of a triangle grid in a diamond pattern.
  - The application and tiling of a texture across the grid
  - Correctly calculated normals
  - A heightmap has been used to generate the heights
  - Noise has been applied to improve realism
- **Rendering of a 3D model**
  - A 3D model can be loaded and rendered
  - The 3D model's textures are applied correctly
- **Lighting and Shading**
  - Models / Terrain are shaded correctly using Phong
  - Directional lighting is working (using Lambert)
  - Point lights
- **Rendering of sky**
  - A skybox is rendered behind other geometry
  - Implemented using a cube map

Scene controls

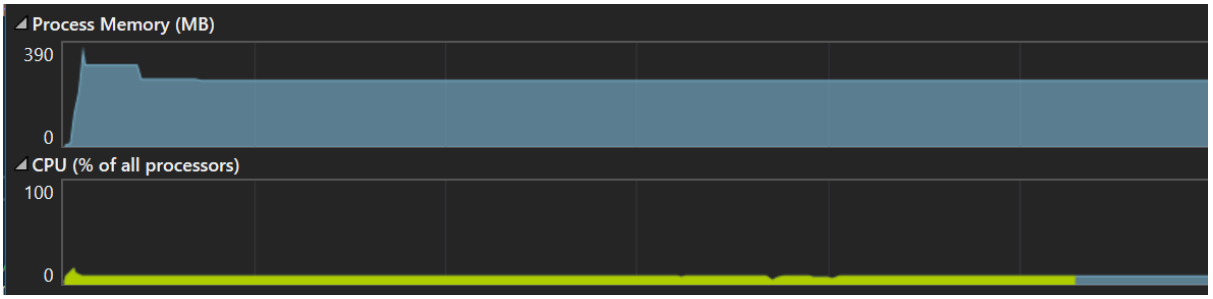
- **Basic controls**
  - W - Moves the camera forward
  - A - Moves the camera to the left
  - S - Moves the camera backwards
  - D - Moves the camera to the right
- **Advanced controls**
  - Left mouse button - Allows for rotation of the camera and movement up and down using the W/S keys based on current direction it is facing
  - Spacebar - Resets the camera to its initial position

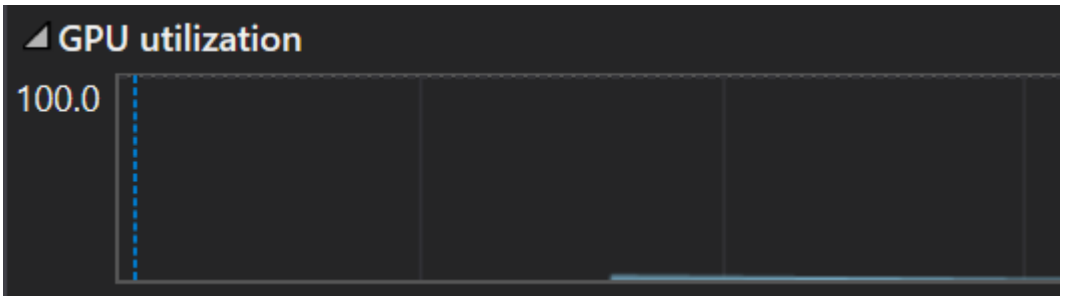
Andrew Richardson W9383619
10/01/2022

Performance

Test system specifications:

- RTX 3070 Ti - 8GB DRR6 VRAM
- AMD Ryzen 5 5600ZX 6-Core processor - 4.2GHz
- 16GB of Corsair DDR4 3600MHz RAM





The process uses a steady 249MB of memory, maxing at during the initial launch at 355MB.
The CPU stays steady at 8% usage and maxes at 15% during the initial launch.
The GPU usages peaks at 1.5% then drops to 0% shortly after

Function Name	Total CPU [unit,...	Self CPU [unit, %]	Module
ThreeGPStart.exe (PID: 19332)	13863 (100.00%)	0 (0.00%)	ThreeGPStart.exe
[External Code]	13863 (100.00%)	11636 (83.94%)	Multiple modules
__scrt_common_main	13603 (98.12%)	0 (0.00%)	ThreeGPStart.exe
__scrt_common_main_seh	13603 (98.12%)	0 (0.00%)	ThreeGPStart.exe
mainCRTStartup	13603 (98.12%)	0 (0.00%)	ThreeGPStart.exe
main	13572 (97.90%)	1 (0.01%)	ThreeGPStart.exe
invoke_main	13572 (97.90%)	0 (0.00%)	ThreeGPStart.exe
Simulation::Update	5157 (37.20%)	5 (0.04%)	ThreeGPStart.exe
Renderer::Render	1911 (13.78%)	21 (0.15%)	ThreeGPStart.exe
ImGui_ImplOpenGL3_RenderDrawData	989 (7.13%)	13 (0.09%)	ThreeGPStart.exe
ImGui_ImplGlfw_NewFrame	798 (5.76%)	2 (0.01%)	ThreeGPStart.exe
Renderer::DefineGUI	774 (5.58%)	5 (0.04%)	ThreeGPStart.exe
ImGui_ImplGlfw_UpdateMouseCursor	668 (4.82%)	2 (0.01%)	ThreeGPStart.exe
ImGui::Begin	645 (4.65%)	68 (0.49%)	ThreeGPStart.exe
ImGui::NewFrame	596 (4.30%)	43 (0.31%)	ThreeGPStart.exe
Helpers::CreateGLFWWindow	561 (4.05%)	0 (0.00%)	ThreeGPStart.exe
Renderer::InitialiseGeometry	463 (3.34%)	29 (0.21%)	ThreeGPStart.exe
Simulation::Initialise	463 (3.34%)	0 (0.00%)	ThreeGPStart.exe

Andrew Richardson W9383619
10/01/2022

The most CPU intensive function aside from the main is the Simulation::Update, however this is not code I have written, so I am unwilling to attempt to modify it as it may break my program if I do.

The next most intensive function after this is the Renderer::Render. This function is one of the most called as it is required to render anything on the screen

```
            670  // Render the scene. Passed the delta time since last called.
            671  void Renderer::Render(const Helpers::Camera& camera, float deltaTime)
  3 (0.02%) 672  {
            673      // Configure pipeline settings
 10 (0.07%) 674      glEnable(GL_DEPTH_TEST);
            675      glEnable(GL_CULL_FACE);
            676
            677      // Wireframe mode controlled by ImGui
  1 (0.01%) 678      if (m_wireframe)
            679          glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
            680      else
 12 (0.09%) 681          glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
            682
            683      // Clear buffers from previous frame
  5 (0.04%) 684      glClearColor(0.0f, 0.0f, 0.0f, 0.f);
488 (3.52%) 685      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
            686
            687      // Compute viewport and projection matrix
            688      GLint viewportSize[4];
 12 (0.09%) 689      glGetIntegerv(GL_VIEWPORT, viewportSize);
  2 (0.01%) 690      const float aspect_ratio = viewportSize[2] / (float)viewportSize[3];
 35 (0.25%) 691      glm::mat4 projection_xform = glm::perspective(glm::radians(45.0f), aspect_ratio, 1.0f, 7500.0f);
            692
            693      // Compute camera view matrix and combine with projection matrix for passing to shader
420 (3.03%) 694      glm::mat4 view_xform = glm::lookAt(camera.GetPosition(), camera.GetPosition() + camera.GetLookVector(), camera.GetUpVector());
            695
            696
            697      // Use our program. Doing this enables the shaders we attached previously.
 66 (0.48%) 698      glUseProgram(skyboxProgram);
            699
            700      // Send the combined matrix to the shader in a uniform
            701      //GLuint combined_xform_id = glGetUniformLocation(m_program, "combined_xform");
            702      //glUniformMatrix4fv(combined_xform_id, 1, GL_FALSE, glm::value_ptr(combined_xform));
            703
  3 (0.02%) 704      glm::mat4 model_xform = glm::mat4(1);
            705
  4 (0.03%) 706      glDepthMask(GL_FALSE);
  3 (0.02%) 707      glDisable(GL_DEPTH_TEST);
            708
            709      //Skybox Render
 11 (0.08%) 710      glm::mat4 view_xform2 = glm::mat4(glm::mat3(view_xform));
 60 (0.43%) 711      glm::mat4 combined_xform2 = projection_xform * view_xform2;
 17 (0.12%) 712      GLuint combined_xform_id2 = glGetUniformLocation(skyboxProgram, "combined_xform_sky");
  7 (0.05%) 713      glUniformMatrix4fv(combined_xform_id2, 1, GL_FALSE, glm::value_ptr(combined_xform2));
            714
  1 (0.01%) 715      glm::mat4 skybox_xform = glm::mat4(1);
  1 (0.01%) 716      GLuint skybox_xform_id = glGetUniformLocation(skyboxProgram, "model_xform");
  1 (0.01%) 717      glUniformMatrix4fv(skybox_xform_id, 1, GL_FALSE, glm::value_ptr(skybox_xform));
            718
  1 (0.01%) 719      GLuint skyboxUniformID = glGetUniformLocation(skyboxProgram, "skybox");
            720
  3 (0.02%) 721      glActiveTexture(GL_TEXTURE0);
 19 (0.14%) 722      glBindTexture(GL_TEXTURE_CUBE_MAP, skyboxMap);
            723
 20 (0.14%) 724      glBindVertexArray(s_VAO);
131 (0.94%) 725      glDrawArrays(GL_TRIANGLES, 0, 36);
            726
            727
  2 (0.01%) 728      glDepthMask(GL_TRUE);
  1 (0.01%) 729      glEnable(GL_DEPTH_TEST);
  4 (0.03%) 730      glBindVertexArray(0);
            731
            732      //Jeep renderer
 29 (0.21%) 733      glUseProgram(jeepProgram);
 40 (0.29%) 734      glm::mat4 combined_xform = projection_xform * view_xform;
  7 (0.05%) 735      GLuint combined_xform_id = glGetUniformLocation(jeepProgram, "combined_xform");
  6 (0.04%) 736      glUniformMatrix4fv(combined_xform_id, 1, GL_FALSE, glm::value_ptr(combined_xform));
            737      glActiveTexture(GL_TEXTURE0);
  5 (0.04%) 738      glBindTexture(GL_TEXTURE_2D, tex);
  8 (0.06%) 739      glUniform1i(glGetUniformLocation(jeepProgram, "sampler_tex"), 0);
```

```
 8 (0.06%)  739     glUniform1i(glGetUniformLocation(jeepProgram, "sampler_tex"), 0);
            740     // Send the model matrix to the shader in a uniform
 1 (0.01%)  741     GLuint model_xform_id = glGetUniformLocation(jeepProgram, "model_xform");
 2 (0.01%)  742     glUniformMatrix4fv(model_xform_id, 1, GL_FALSE, glm::value_ptr(model_xform));
            743     //// Bind our VAO and render
 7 (0.05%)  744     glBindVertexArray(j_VAO);
70 (0.50%)  745     glDrawElements(GL_TRIANGLES, j_numElements, GL_UNSIGNED_INT, (void*)0);
 7 (0.05%)  746     glBindVertexArray(0);
            747
            748     //Terrain renderer
14 (0.10%)  749     glUseProgram(terrainProgram);
40 (0.29%)  750     glm::mat4 terrain_combined_xform = projection_xform * view_xform;
 6 (0.04%)  751     GLuint terrain_combined_xform_id = glGetUniformLocation(terrainProgram, "combined_xform");
 3 (0.02%)  752     glUniformMatrix4fv(terrain_combined_xform_id, 1, GL_FALSE, glm::value_ptr(terrain_combined_xform));
            753     glActiveTexture(GL_TEXTURE0);
            754     glBindTexture(GL_TEXTURE_2D, t_tex);
 1 (0.01%)  755     glUniform1i(glGetUniformLocation(terrainProgram, "sampler_tex"), 0);
 4 (0.03%)  756     model_xform = glm::mat4(1);
 2 (0.01%)  757     GLuint terrain_model_xform_id = glGetUniformLocation(terrainProgram, "model_xform");
            758     glUniformMatrix4fv(terrain_model_xform_id, 1, GL_FALSE, glm::value_ptr(model_xform));
 8 (0.06%)  759     glBindVertexArray(t_VAO);
57 (0.41%)  760     glDrawElements(GL_TRIANGLES, m_numElements, GL_UNSIGNED_INT, (void*)0);
 4 (0.03%)  761     glBindVertexArray(0);
            762
            763     //Cube renderer
20 (0.14%)  764     glUseProgram(cubeProgram);
66 (0.48%)  765     combined_xform = projection_xform * view_xform;
            766
 1 (0.01%)  767     combined_xform_id = glGetUniformLocation(cubeProgram, "combined_xform");
 1 (0.01%)  768     glUniformMatrix4fv(combined_xform_id, 1, GL_FALSE, glm::value_ptr(combined_xform));
            769
 1 (0.01%)  770     model_xform = glm::mat4(1);
17 (0.12%)  771     model_xform = glm::translate(model_xform, glm::vec3{ 1000.0f, 500.0f, 500.0f });
22 (0.16%)  772     model_xform = glm::scale(model_xform, glm::vec3{ 10.0f, 10.0f, 10.0f });
            773
            774     //// Uncomment all the lines below to rotate cube first round y then round x
            775     static float angle = 0;
            776     static bool rotateY = true;
            777
            778     if (rotateY) // Rotate around y axis
30 (0.22%)  779         model_xform = glm::rotate(model_xform, angle, glm::vec3{ 0 ,1,0 });
            780     else // Rotate around x axis
32 (0.23%)  781         model_xform = glm::rotate(model_xform, angle, glm::vec3{ 1 ,0,0 });
            782
            783     angle += 0.001f;
            784     if (angle > glm::two_pi<float>())
            785     {
            786         angle = 0;
            787         rotateY = !rotateY;
            788     }
            789
 4 (0.03%)  790     model_xform_id = glGetUniformLocation(cubeProgram, "model_xform");
 5 (0.04%)  791     glUniformMatrix4fv(model_xform_id, 1, GL_FALSE, glm::value_ptr(model_xform));
            792
 4 (0.03%)  793     glBindVertexArray(c_VAO);
37 (0.27%)  794     glDrawElements(GL_TRIANGLES, c_numElements, GL_UNSIGNED_INT, (void*)0);
 5 (0.04%)  795     glBindVertexArray(0);
            796
 2 (0.01%)  797 }
            798
            799
```

The most intensive parts of this code are the clearing of the screen and the camera view matrix. The camera view matrix accounted for 3.03% of CPU usage.
glClear combined with glClearColor account for 3.56% of CPU usage between them, by disabling the glClearColor function, the CPU usage is reduced by 0.63%, to 2.93%, which may not seem like a large amount but this could have a large impact on performance in a larger program. This small change also had the unexpected effect of lowering the CPU usage of the camera view matrix by 0.49%, bringing it to 2.54% overall. These changes can be seen by examining the screenshots above and below.

```
            683     // Clear buffers from previous frame
            684     //glClearColor(0.0f, 0.0f, 0.0f, 0.f);
233 (2.93%) 685     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
            686
            687     // Compute viewport and projection matrix
```

```
202 (2.54%)  693     // Compute camera view matrix and combine with projection matrix for passing to shader
             694     glm::mat4 view_xform = glm::lookAt(camera.GetPosition(), camera.GetPosition() + camera.GetLookVector(), camera.GetUpVector());
             695
```

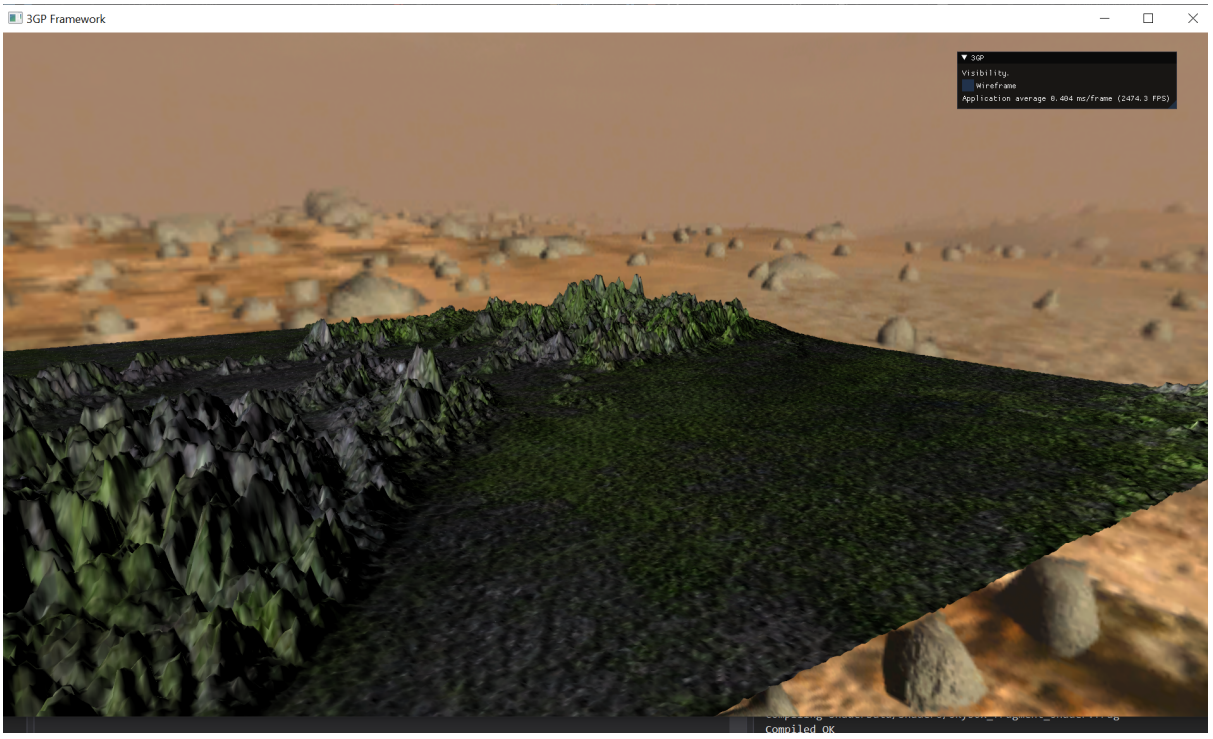
glClearColor can be safely disabled from the code as its functionality is essentially overridden by the presence of the skybox, so any colour it would clear to is unable to be seen.

This change also had smaller effects on other functions, decreasing the CPU usages of a number of them while increasing usage on other sections. Overall this still had a positive effect on the CPU usage of the program and some of the fluctuations noticed in repeated testing seemed to be caused by various background processes of the system used for testing

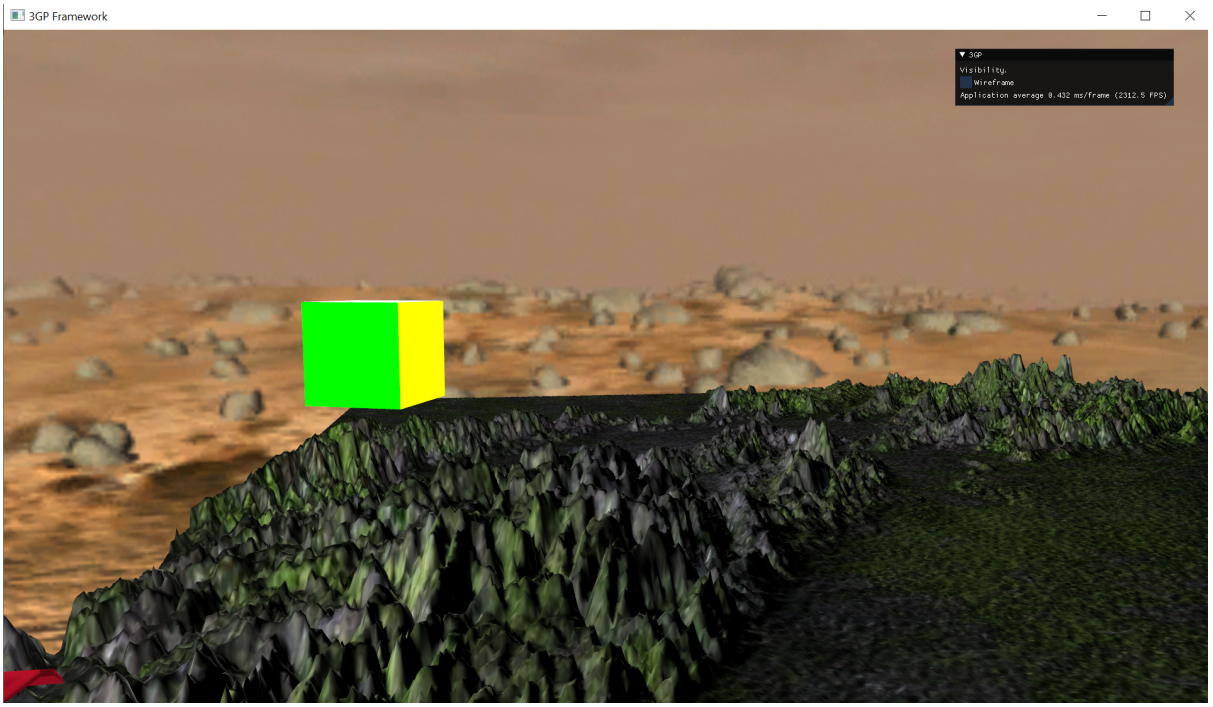
Future Implementations

I would like to add a dynamic skybox that rotates through a day/night cycle to add a sense of realism to the scene, as well as implementing a repeating texture to the terrain to prevent the texture appearing stretched out.

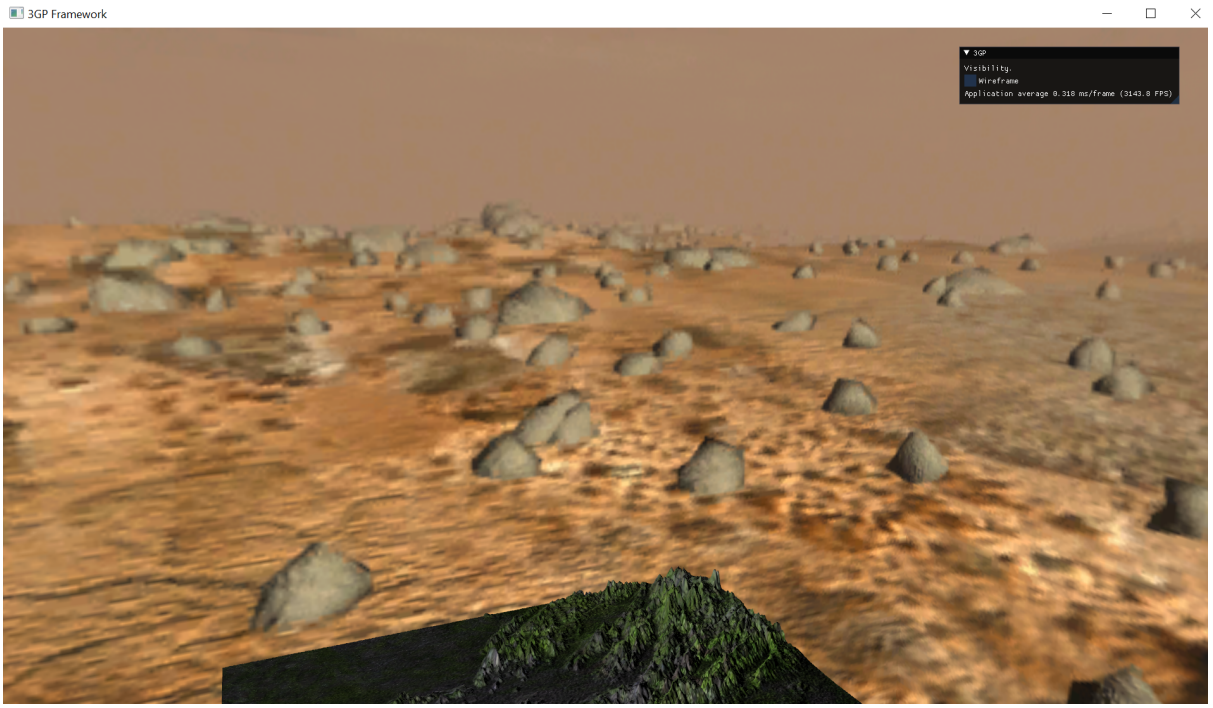
Screenshots

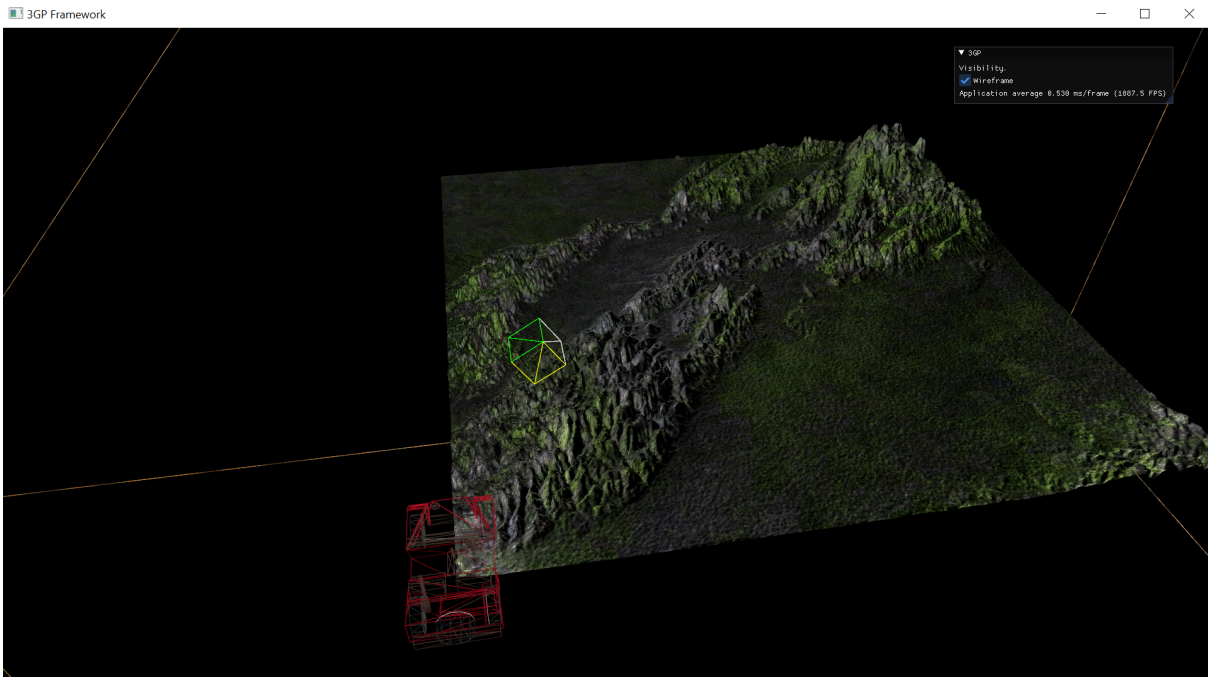


Andrew Richardson W9383619
10/01/2022



Andrew Richardson W9383619
10/01/2022





Andrew Richardson W9383619
10/01/2022