Проект по Компјутерска Графика

Генерирање релјеф од 2D слика во OpenGL

Виктор Младеновски (216151)

Содржина и цели

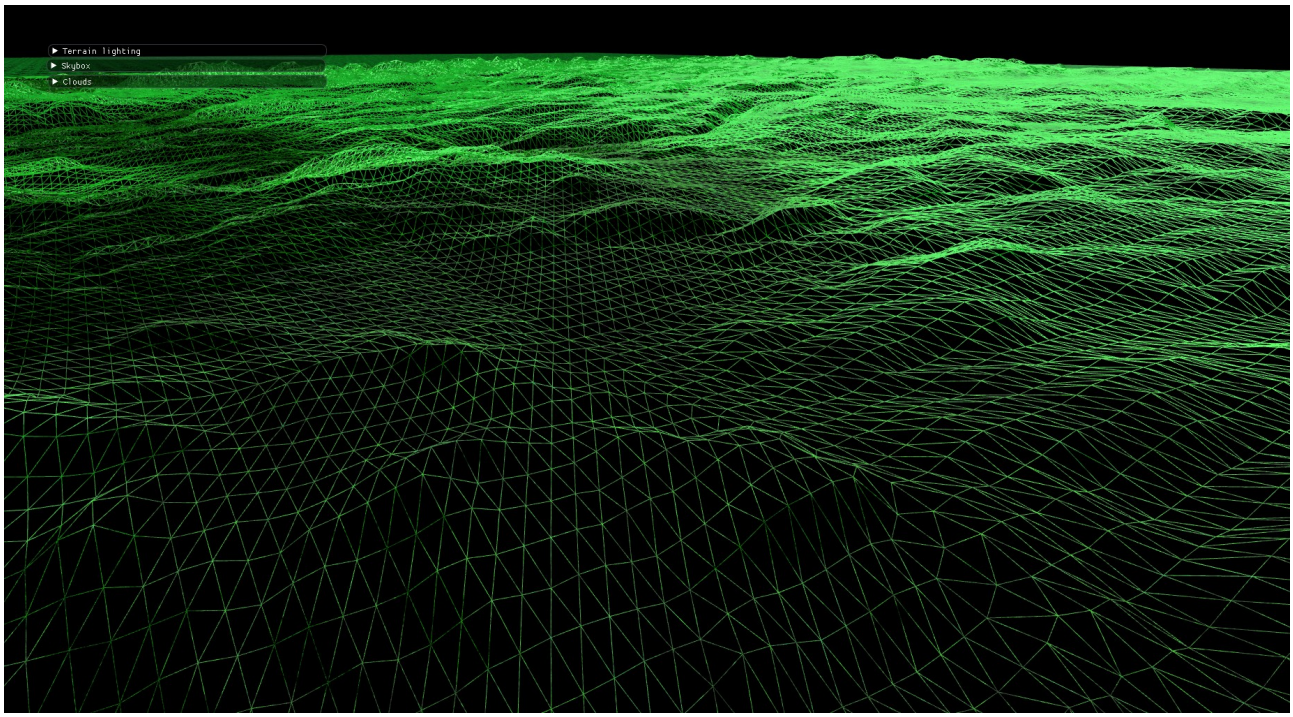
Проектот во најголем дел опфаќа реимплементација на метод[1] за рендерирање на релјеф составен од тродимензионална мрежа од точки, со помош на дводимензионална текстура (heightmap), чијшто тексели одговараат на „висината“ на точките. Друга цел и предизвик беше ваквиот пристап да се вклопи во создавање на што е можно пореална сцена со помош на графичката библиотека OpenGL. Тоа претежно вклучува сенчање на објектите во сцената т.е. примена на светлинскиот модел и нивно текстурирање, како и рендерирање на облаци (тродимензионални, кои не се дел од skybox-от на сцената).

Рендерирање на мрежата

Слика 1: src/terrainmaps/heightmap.png - текстурата користена за проектот, 3840 x 1910 px

1 Статиите https://learnopengl.com/Guest-Articles/2021/Tessellation/Height-map и https://learnopengl.com/Guest-Articles/2021/Tessellation/Tessellation од Dr. Jeffrey Paone.

Наивниот пристап за ова вклучува изминување на секој тексел и пресметување на локалните координати на соодветната точка на CPU страната, пред истата да се смести во framebuffer. Имено, *x* и *z* координатите на точката се пресметуваат како функции од позицијата на текселот, а *y* координатата како функција од неговите RGBA својства. Мрежата потоа може да се исцрта со помош на ленти од триаголници, пр. со *GL_TRIANGLE_STRIP* (значи теселирана пред праќање до GPU):

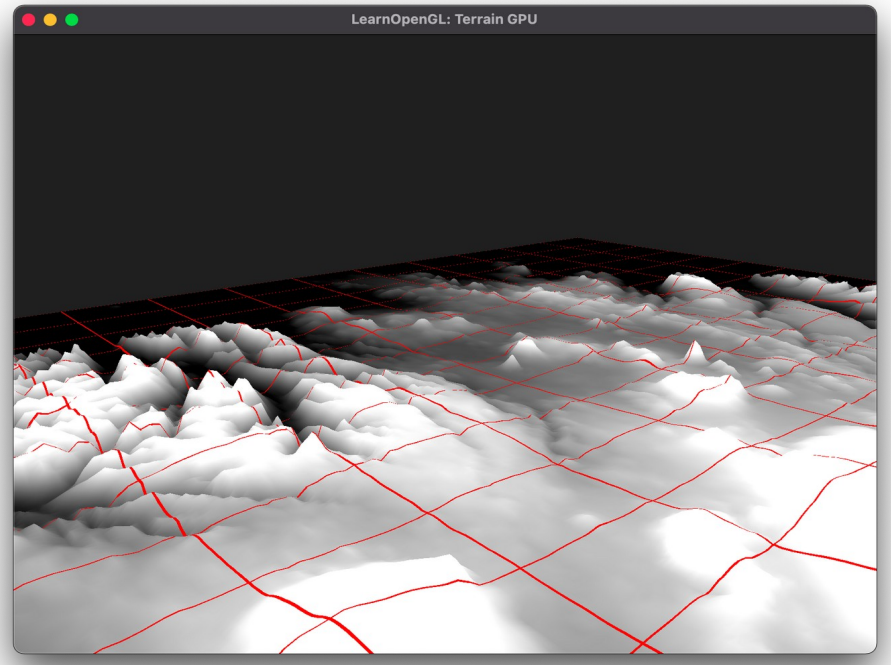


Слика 2

Овој пристап има неколку недостатоци, како на пример квадратната часовна и мемориска комплексност, кои се пропорционални со резолуцијата на текстурата, што може да биде огромна ако се бараат поквалитетни резултати. Доколку се посматраат перформансите на хардверот на кој се извршува програмата, на пример преку мерење на температурата, може да се забележи дека и графичкиот процесор е мошне оптоварен, многу повеќе споредено со вториот пристап. Објаснување за тоа може да биде фактот дека vertex шејдерот конверзијата во нормализирани координати на уредот (пресметката *Proj * View * Model * **pos***) ја прави подеднакво за фиксен број темиња – *w * h*, ако *w* и *h* се димензиите на текстурата.

Еден начин ова да се оптимизира е да се користат точни пресметки само за некои темиња, со чија помош ќе се интерполираат вредностите за останатите. Пресметките, покрај множењето споменато погоре, може да вклучуваат и одредување на висината на исходната точка на

релјефот, така што текстурата ќе се семплира само на избрани темиња. Токму ова го прави подобриот пристап (детално објаснет во втората статија), кој избира одредени позиции од текстурата и ги предава на специлизиран шејдер за теселација што ги интерпретира[2] како темиња на четириаголници за чијшто внатрешни точки треба да се интерполира.



Слика 3, извор learnopengl.com

Сепак, вака перформансите најмногу ќе зависат од избраниот број на внатрешни темиња на секој четириаголник. Всушност, втората техника станува скоро еквивалентна во однос на перформански со втората, доколку се избере доволно голем број на patch-ови и степен на теселација, односно број на внатрешни точки по patch. Конкретно, ако степенот на теселација (комбинација од неколку променливи) се фиксира на 64,

```
gl_TessLevelOuter[0] = 64;
gl_TessLevelOuter[1] = 64;
gl_TessLevelOuter[2] = 64;
gl_TessLevelOuter[3] = 64;
gl_TessLevelInner[0] = 64;
gl_TessLevelInner[1] = 64;
```

(src/shaders/tessellation_control.shader)

2 Се праќаат со наредбата *glDrawArrays(GL_PATCHES, first, count).* Примитивите може да бидат произволни многуаголници, во случајов се четириаголници бидејќи така е наведено со *glPatchParameteri(GL_PATCH_VERTICES, 4).*

и ако се користат 42^2 четириаголници,

```
unsigned rez = 42;
```
(*src/main.cpp*)

бидејќи $64^2 * 42^2 \approx 3840 * 1910$, кодот ќе има скоро исти перформанси како со употребата на првиот пристап. Премали вредности за параметрите, секако, го нарушуваат квалитетот на сцената бидејќи се исцртуваат премалку триаголници.

За да вистина се постигне поголема ефикасност, а да се минимизираат загубите на квалитет во крајниот исход, алгоритмот степенот на теселација го одредува динамички[3] за секој patch т.е. врз основа на растојанието на неговите темиња од позицијата на погледот. За четириаголниците поблиску до погледот се обработуваат повеќе внатрешни точки, а за подалечните помалку.

Овој процес практично се реализира со употреба на три посебни шејдери: Tessellation Control Shader – пресметува растојание од темињата на patch-от до позицијата на погледот и со тоа го одредува нивото на теселација; Tessellation Primitive Generator – ја врши поделбата страниците на patch-от и ги одредува позициите на внатрешните точки и Tessellation Evaluation Shader каде првин се семплира текстурата на за точките што се добиваат на влез и се определува висиниата, а потоа се врши конверзијата во координати на уред.

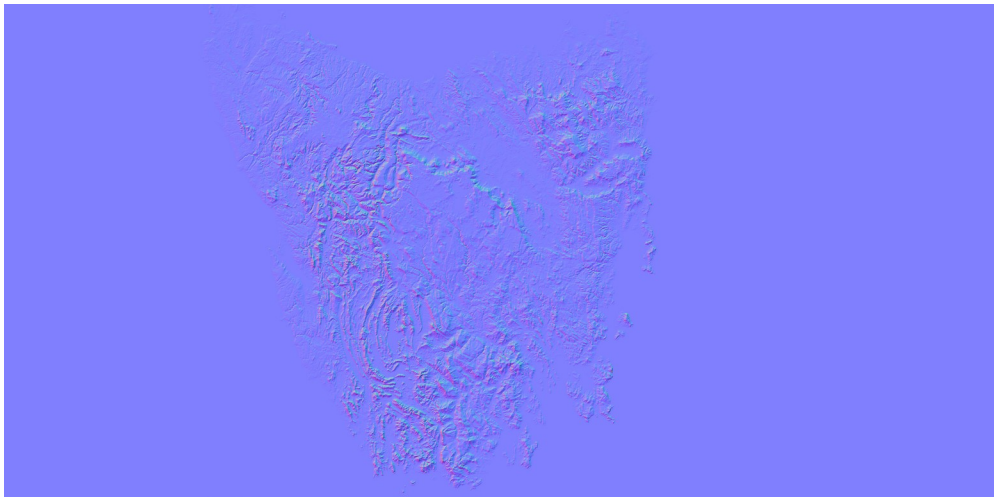
Сенчање

Пресметаната висина која се предава на fragment шејдерот може да се искористи за некое едноставно сенчање на релјефот, но недоволно е за пореалистична сцена. За таа цел се потребни векторите нормални на површината (за секој триаголник). Во имплементацијата на *learnopengl.com* еден од шејдерите за теселација истите ги пресметува:

```
vec4 normal = normalize( vec4(cross(vVec.xyz, uVec.xyz), 0) );
```
(*src/tessellation_eval.shader*)

3 Техникава се среќава како Dynamic LOD (Level Of Detail) и е главната придобивка од вршење теселација на GPU страната.

но, доколку овој вектор се предаде на fragment шејдерот, не се однесува како што е очекувано и дава чудни резултати. Како „workaround“ за добивање на нормалите, во проектот се користи мапа на нормали, односно текстура генерирана со помош на онлајн алатка:

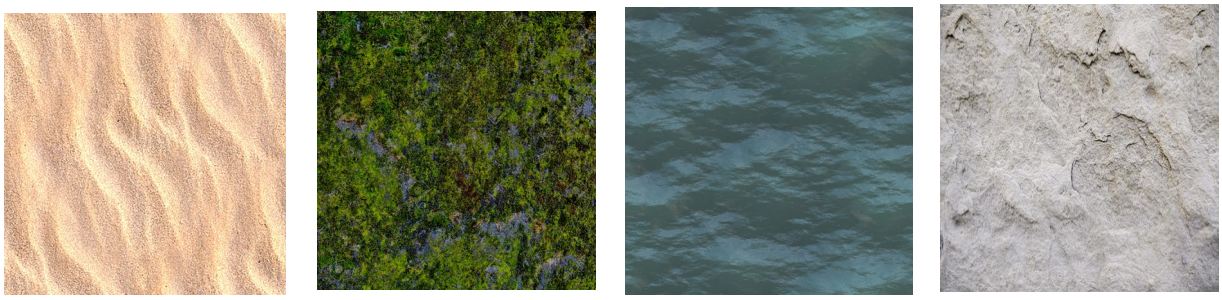


Слика 4: src/terrainmaps/normalmap.png

```
vec3 normal = texture(normalMap, texCoord).rgb;
    normal = normalize(normal * 2.0 - 1.0);
```
(*src/fragment.shader*)

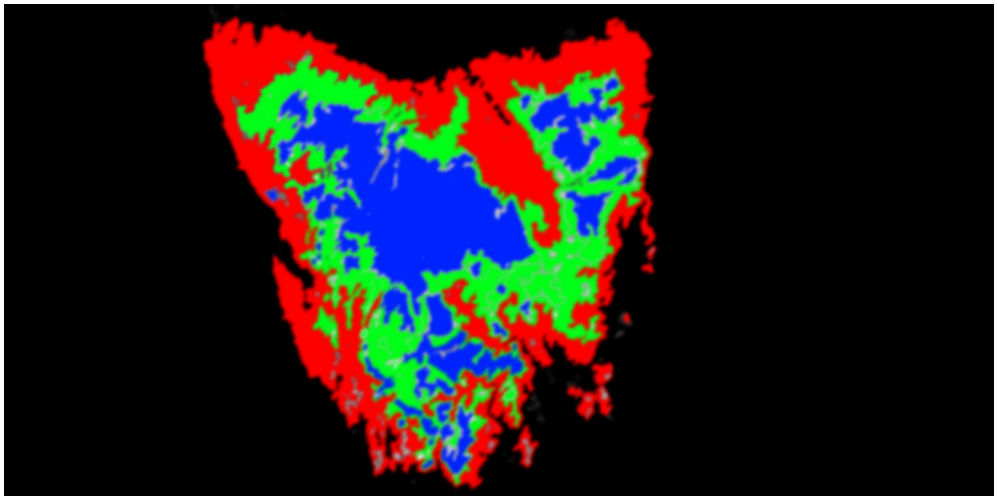
Ова овозможува целосно да се примени светлинскиот модел т.е. понатаму да се пресметуваат огледалната и дифузната рефлексија.

За текстурирање на површината се користат слики од 4 типови материјали – песок, мов, вода и камен.



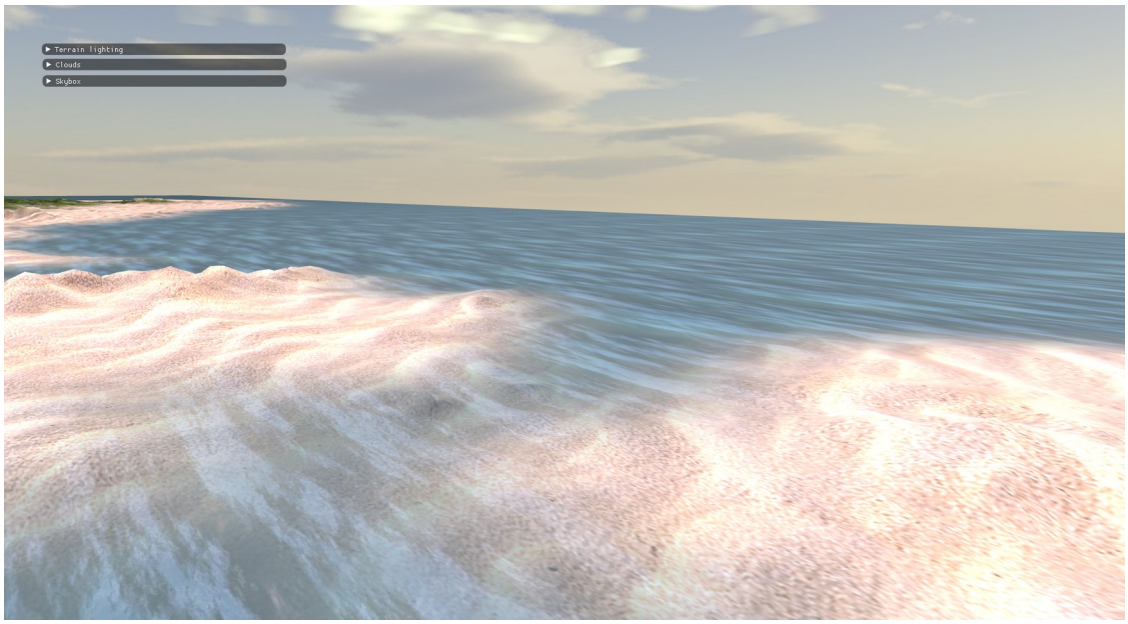
Слика 5: src/textures

На кои делови од површината да се применат текстурите се одлучува преку семплирање на посебна текстура – blend мапа:



Слика 6: src/terrainmaps/textureblendmap.png

Зависно од RGB својствата на секој тексел, се одредува уделот на секоја од четирите текстури во конечниот резултат. Во конкретниов случај, црвената боја одговара на песокот, зелената на мовта, сината на каменестата текстура, додека количеството на текстурата за вода се пресметува како разликата *1 – (blendmap.R + blendmap.G + blendmap.B)* т.е. одговара на отсуството на останатите три текстури. За да се избегнат остри граници помеѓу различно текстурираните површини и за да се добие ефект на прелевање од една во друга текстура (сл. 7), врз blend мапата е применет Гаусов blur (иако истиов ефект, веројатно, може да се постигне и само со GLSL).



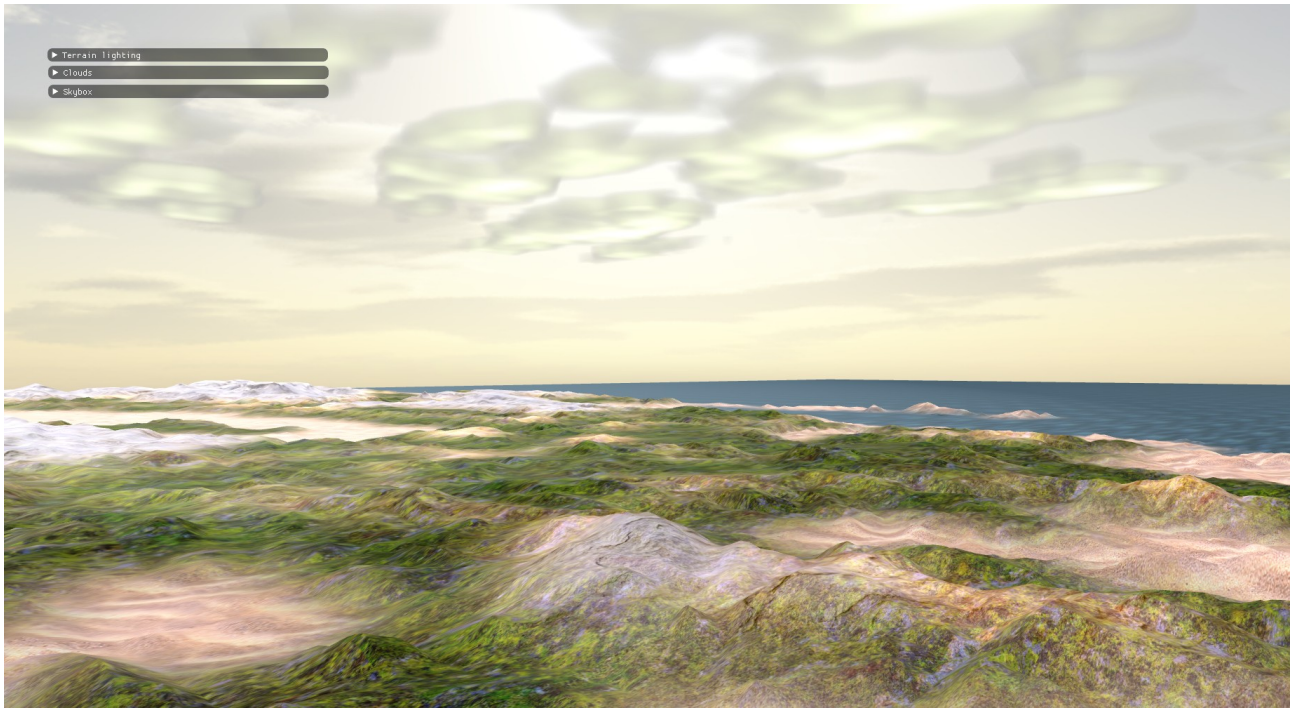
Слика 7

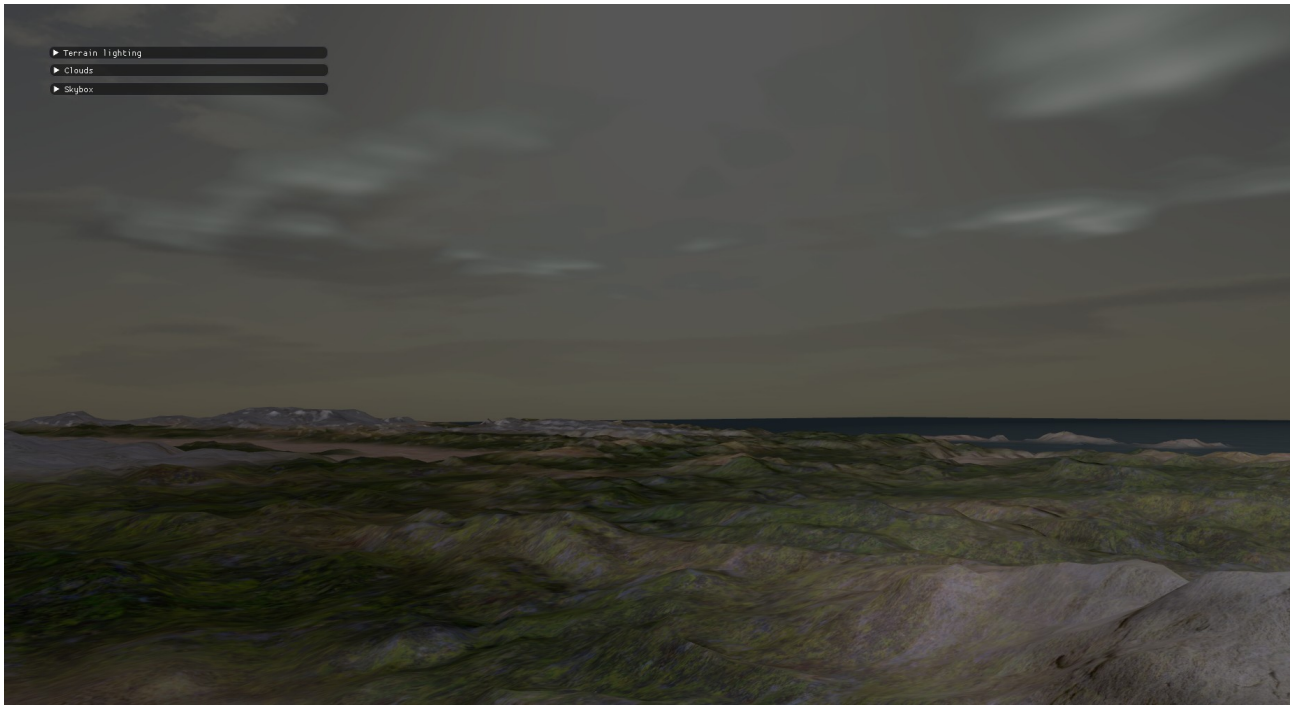
Рендерирање облаци

Изворниот код содржи едноставна техника за исцртување на тродимензионални облаци. Не се работи за volume ray marching[4], иако тоа беше првичната цел. Се цртаат 33 правоаголници на мало вертикално растојание, на кои потоа се наметнува текстура генерирана во самиот shader со помош на Perlin шум. Ваквиот пристап не е најпаметен во однос на перформанси, бидејќи бара додатни пресметки од GPU-то, но дава повеќе контрола врз изгледот на облаците (параметри како густина, големина, распространетост итн.).

Слики од крајната верзија

4 Посложена техника која вклучува пуштање зраци за секој пиксел од сликата низ волумен (3D текстура) и семплирање на текстурата на точки кои лежат на патеката на зракот. Најсоодветна за проекти каде камерата голем дел од времето е внатре во облаците, пр. симулатори на лет.


Terrain lighting
Clouds
Skybox


Terrain lighting
Clouds
Skybox

Користена литература, извори на код и алатки:

1. https://learnopengl.com/Guest-Articles/2021/Tessellation/Height-map
2. https://learnopengl.com/Guest-Articles/2021/Tessellation/Tessellation
3. https://www.reddit.com/r/opengl/comments/g2dr40/yet_another_problem_with_raymarching_volume/
4. https://www.guerrilla-games.com/media/News/Files/The-Real-time-Volumetric-Cloudscapes-of-Horizon-Zero-Dawn.pdf
5. https://www.youtube.com/watch?v=-kbal7aGUpk
6. https://github.com/joksim/OpenGLPrj
7. https://github.com/ocornut/imgui
8. https://tangrams.github.io/heightmapper/
9. https://cpetry.github.io/NormalMap-Online/