

**PROIECT PENTRU OBŢINEREA ATESTĂRII PROFESIONALE ÎN INFORMATICĂ**

**GENERARE TEREN**

**Cuprins**

1. Alegerea temei
2. Generalităţi despre C++, OpenGL, GLUT si Wavefront .obj
3. Algoritmul de Perlin Noise
4. Structura, conţinutul şi implementarea proiectului
5. Indicaţii de utilizare
6. Bibliografie

**Alegerea temei**

Fiind pasionat de lucrurile ce se intâmplă aleatoriu am decis că ar fi interesant să vad cum funcționează generarea de teren și să încerc să implementez un soft ce generează suprafețe. Inițial proiectul era cu scopul de studiu asupra algoritmului de generare de hărți bazate pe așa zisul „noise”, dar către final am zis să studiez puțin și despre fișierele de tip .obj folosite în modelarea 3d. Mi s-a părut că o opțiunea de a salva harta generată nu necesită multă muncă pentru a fi implementată, așa că am adăugat și această funcție.

Am ales să lucrez în limbajul C++ deoarece voiam ca aplicația mea sa fie cât mai aproape de hardware și nu voiam biblioteci/softuri complexe precum Unity, Unreal, C# etc. De asemenea nu voiam să o implementez pentru web folosind Javascript. Făcând cunoștiință cu OpenGL în vara din 2017 am zis că ar fi cea mai ușoară soluție pentru mine deoarece bazele in C++ le aveam de la școală, iar funcțiile din biblioteca GLUT îmi deveneau familiare. Deși GLUT este o biblioteca ceva mai veche (acum există FreeGLUT, e mai nouă) am zis că nu are rost să mă complic și să încerc să învăț de la 0 o bibliotecă nouă.

Pentru generare am folosit algoritmul de Perlin Noise deoarece acesta este mai ușor de implementat și dă rezultate realiste (comparabil cu Value Noise sau Simplex Noise).

Pentru salvarea modelului 3d am folosit fișiere de tip Wavefront .obj deoarece sunt open source iar exportarea e ușor de implementat și nu necesită biblioteci externe.

**Generalități**

**C++**

C++ este un limbaj de programare general, compilat. Este un limbaj multi-paradigmă, cu verificarea statică a tipului variabilelor ce suportă programare procedurală, abstractizare a datelor, programare orientată pe obiecte. În anii 1990, C++ a devenit unul din cele mai populare limbaje de programare comerciale, rămânând astfel până azi.

Bjarne Stroustrup de la Bell Labs a dezvoltat C++ (inițial denumit C cu clase) în anii 1980, ca o serie de îmbunătățiri ale limbajului C. Acestea au început cu adăugarea noțiunii de clase, apoi de funcții virtuale, suprascrierea operatorilor, moștenire multiplă, șabloane (engleză template) și excepții.

**OpenGL**

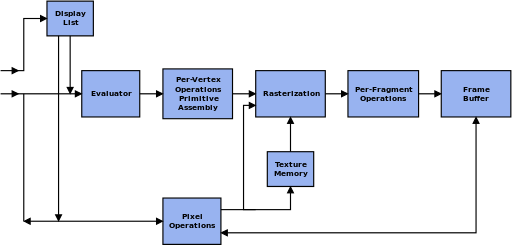
OpenGL (Open Graphics Library) este o specificație a unui standard care definește un API (Application Programming Interface) multiplatformă foarte utilizat pentru programarea componentelor grafice 2D și 3D ale programelor de calculator. Interfața constă în peste 250 de apeluri diferite care folosesc la a desena pe ecranul calculatorului scene 3D complexe din primitive (din primitives, elemente simple). OpenGL a fost inițial dezvoltat de compania Silicon Graphics, Inc. (SGI) în 1992 și este foarte utilizat în grafică asistată de calculator, realitate virtuală, vizualizare științifică, simulări de zboruri sau jocuri pe calculator. Acest ultim domeniu este în strânsă competiție cu tehnologia DirectX de la Microsoft (compară OpenGL cu Direct3D). Proiectul OpenGL este condus de compania Khronos Group, un consorțiu tehnologic non-profit.

**GLUT**

În prezent există implementări OpenGL pentru sistemele de operare Microsoft Windows, Unix şi IBM PS/2. Acestea se prezintă sub forma unor biblioteci integrate (sau care pot fi integrate) în mediile de dezvoltare a aplicaţiilor. Deoarece OpenGL nu conţine funcţii de gestiune a ferestrelor de afişare şi interacţiune cu utilizatorul, implementările OpenGL sunt completate cu astfel de funcţii. O asemenea implementare extinsă este şi GLUT (OpenGL Utilities Toolkit).

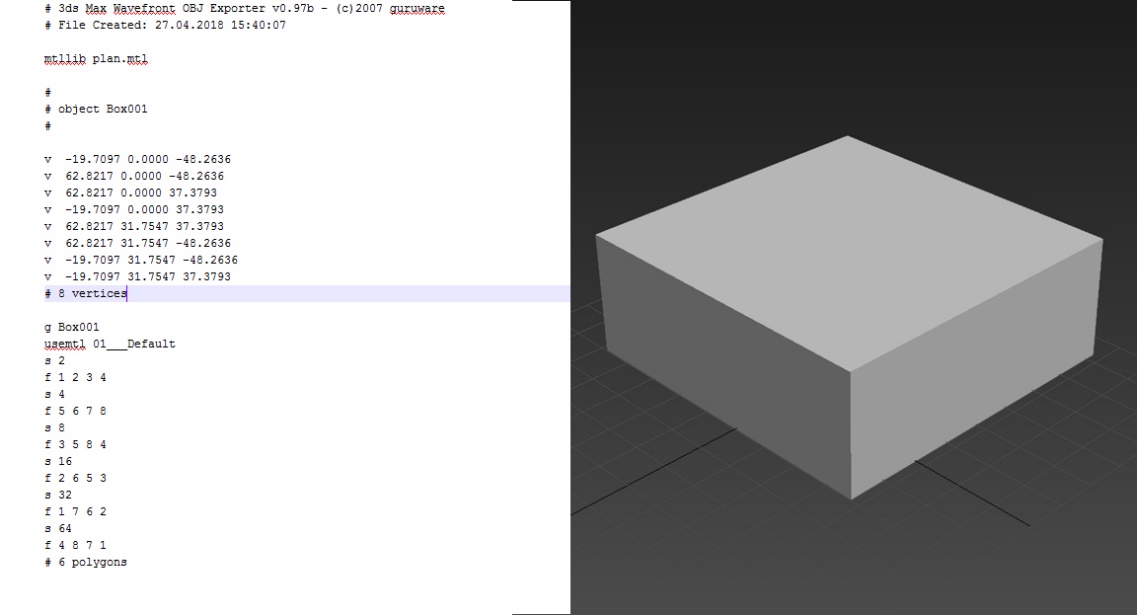
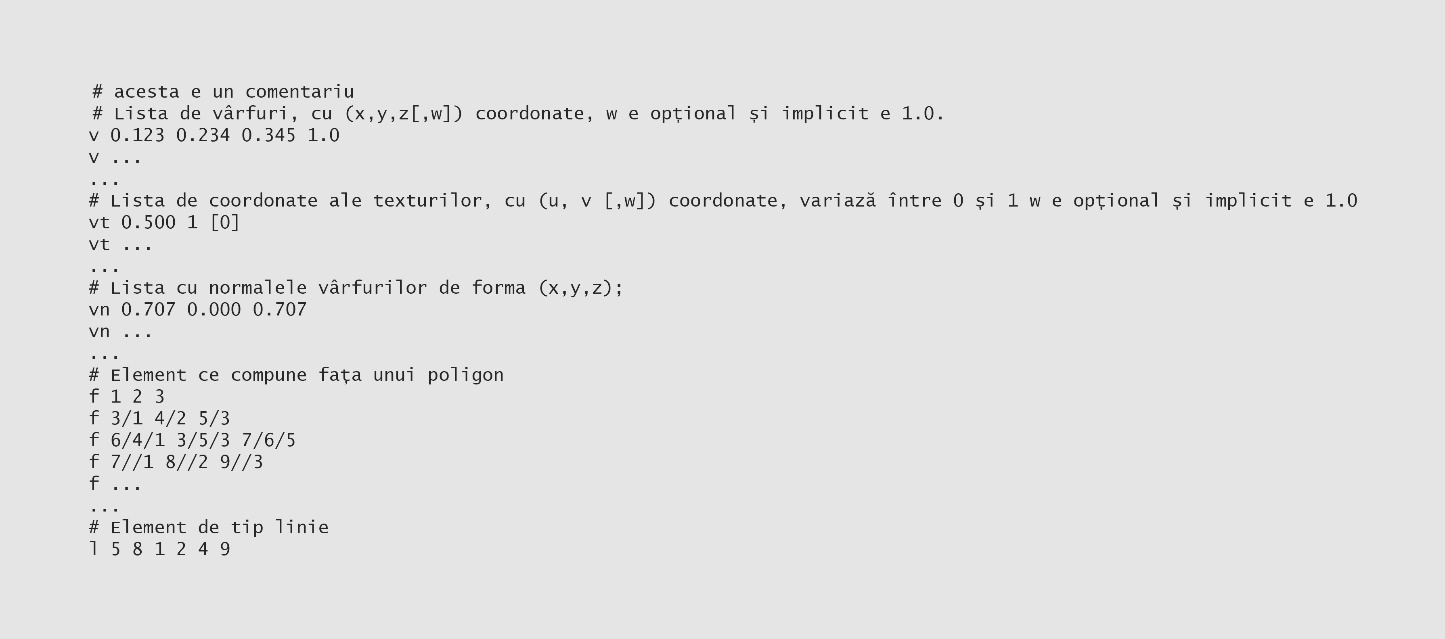
Dintre funcţiile de sinteză de nivel înalt oferite de OpenGL menţionăm:

* Maparea texturilor: aplicarea de imagini pe suprafeţe 3D
* Eliminarea automată din imagini a părţilor nevizibile ale obiectelor prin algoritmul Z-buffer
* Efecte de iluminare a scenelor 3D folosind diferite modele de iluminare şi una sau mai multe surse de lumină
* Simularea reflexiei și a transmisiei luminii ţinând cont de proprietăţile materialelor
* Transformarea din spaţiul 3D utilizator în spaţiul 2D ecran prin specificarea matricelor de transformare, posibilitatea de a modifica poziţia şi dimensiunea obiectelor în spaţiul 3D utilizator



**Wavefront .obj**

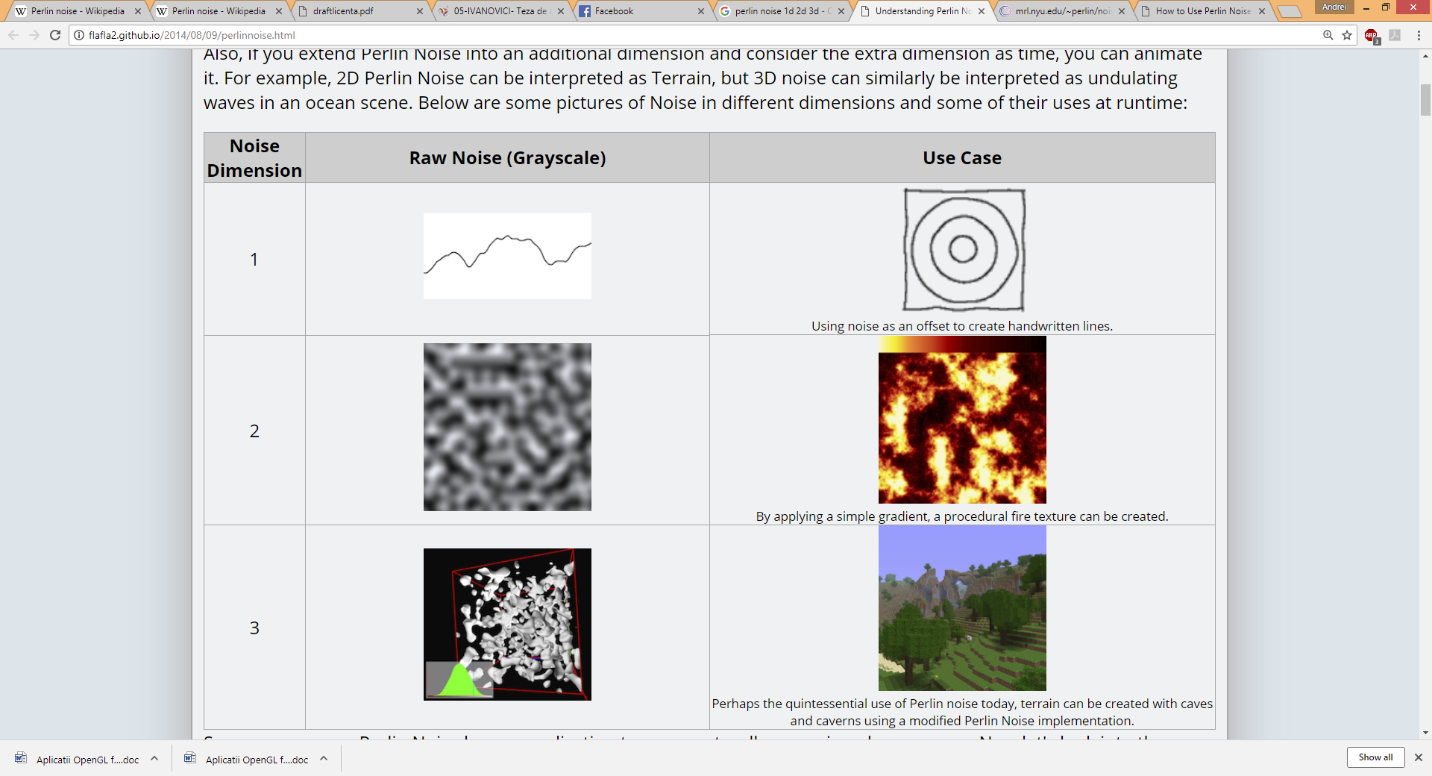
Fișierele OBJ sunt fișiere folosite pentru definirea geometrică și au fost introduse de Wavefront Technologies pentru pachetul lor de animație numit Advanced Visualizer. Acesta este un tip de fișier open source și a fost adoptat de o multitudine de alte aplicații. Formatul fișierelor este relativ simplu deoarece ascesta conține doar informații despre geometria 3D a poligoanelor și anume poziția vârfurilor, coorodnate UV, fețele ce compun poligoanele etc. Vârfurile sunt definite în sens trigonometric așa că declararea normalelor nu e neapărată. Fișierele OBJ nu conțin informații despre unități de măsură dar informații despre dimensiune pot fi scrise în comentarii pentru a putea fi citite de utilizator.

****Exemplu de fișier:

**Algoritmul de Perlin Noise**

Perlin Noise este o metodă de generare a zgomotului gândită și implementată pentru prima dată de Ken Perlin în anul 1983. Motivul pentru care acesta a decis să dezvolte acest procedeu este faptul că la momentul respectiv grafica pe calculator avea un aspect mult prea mecanic, sau sintetic (“machine­like”). În prezent, această tehnică este folosită preponderent în industria efectelor vizuale, mulțumită gradului de realism pe care aceasta o produce. Folosind Perlin Noise, se generează suprafețe de obiecte, foc, fum și nori cu un aspect realistic, imitând apariția aparent aleatorie a acestora în natură.

Definiția formală a acestuia, după cum a formulat­o Matt Zucker în lucrarea sa “Perlin Noise Math FAQ” (2001) este următoarea: Perlin Noise este o funcție folosită pentru generarea unui zgomot coerent peste un spațiu, în condițiile în care coerența zgomotului reprezintă faptul că pentru oricare două puncte din acel spațiu valoarea funcției se schimbă cu finețe trecând de la un punct la celălalt, fără să existe discontinuități.

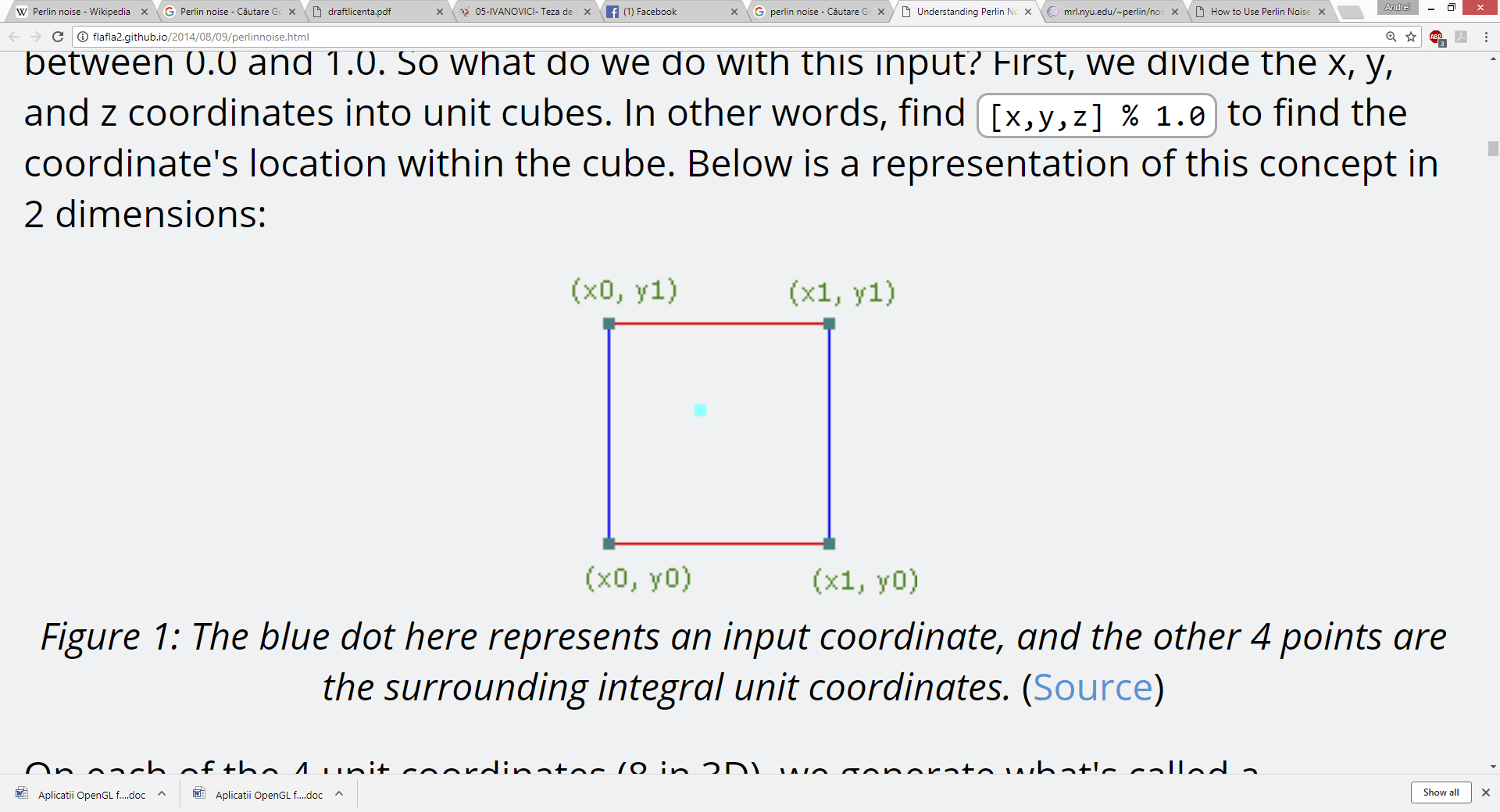
Ce reprezintă, de fapt, o funcție de zgomot este o funcție care pentru niște coordonate într­un spațiu furnizează un număr real cu valoarea cuprinsă între ­1 și 1. Aceste funcții pot fi aplicate unui spațiu, indiferent de numărul acestuia de dimensiuni.

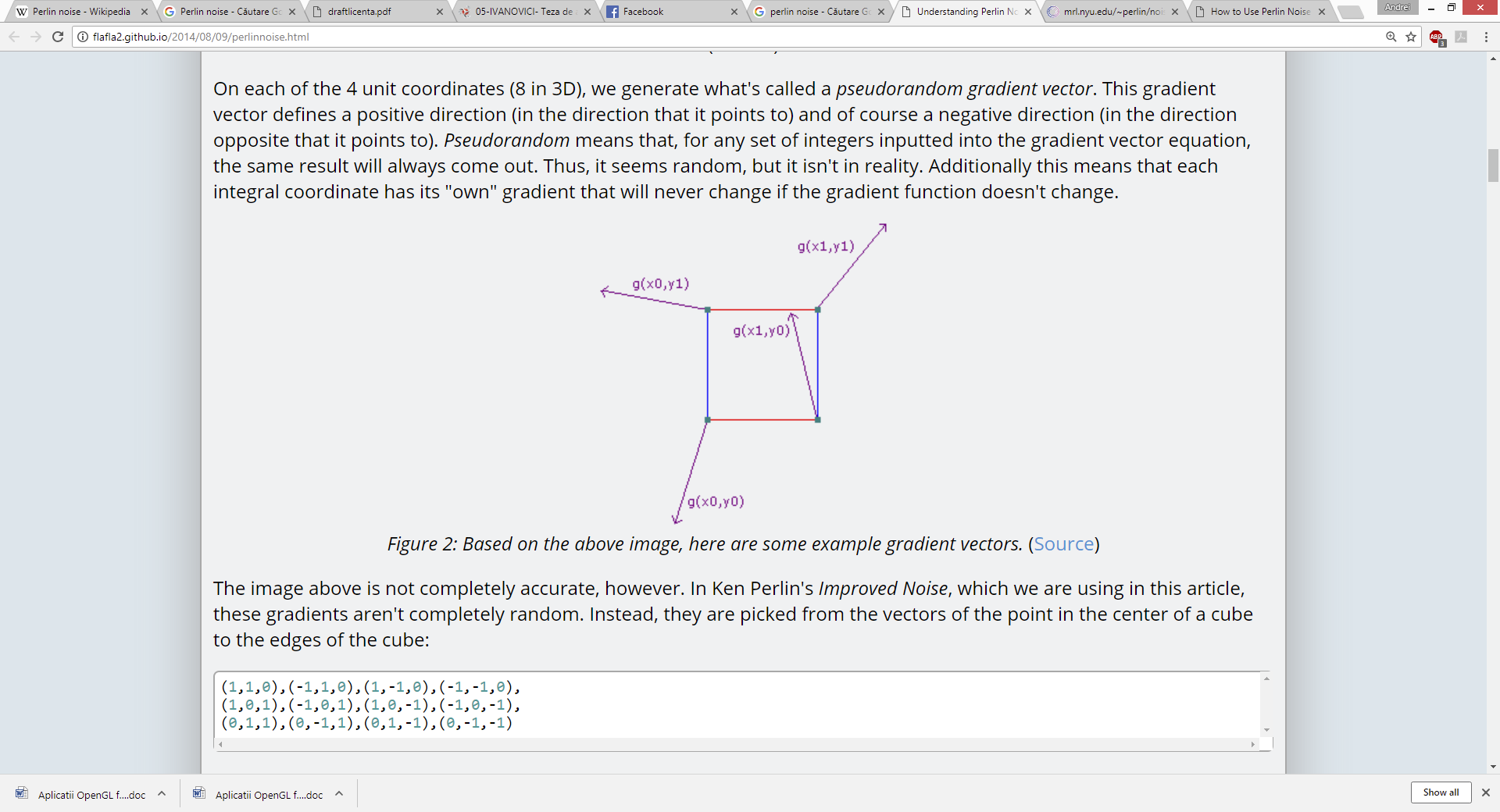
Astfel, pe lângă aplicarea 3D pe care am utilizat­o în acest proiect, pot fi implementate versiuni și pentru o singură dimensiune (rezultatul va avea forma unui grafic) sau două dimensiuni.

În cele ce urmează, voi descrie generarea de Perlin Noise după modelul explicației lui Matt Zucker în “The Perlin Noise math FAQ”

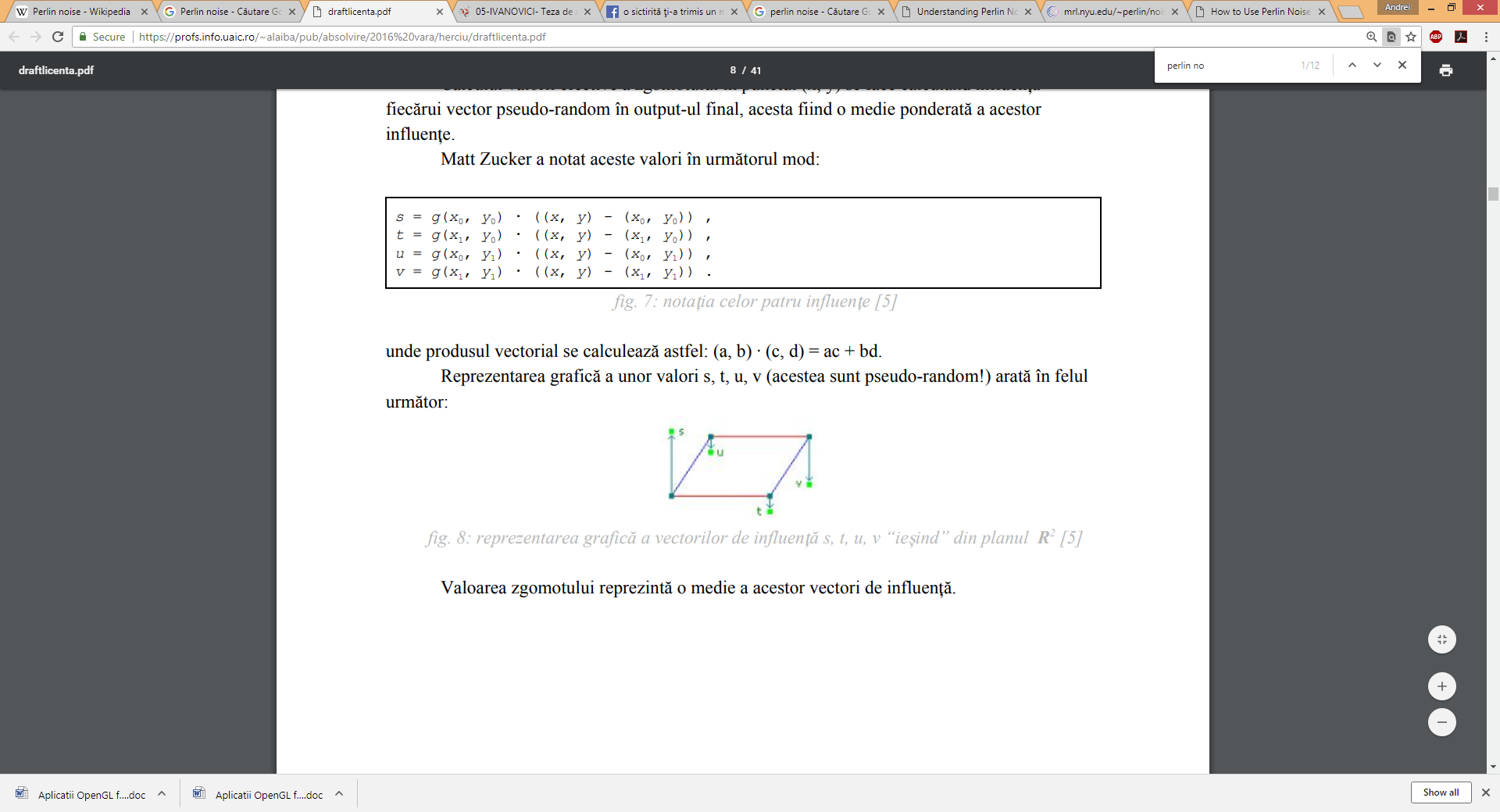
În cazul 2D, funcția va arăta în felul acesta: **noise2d(x, y) = z**

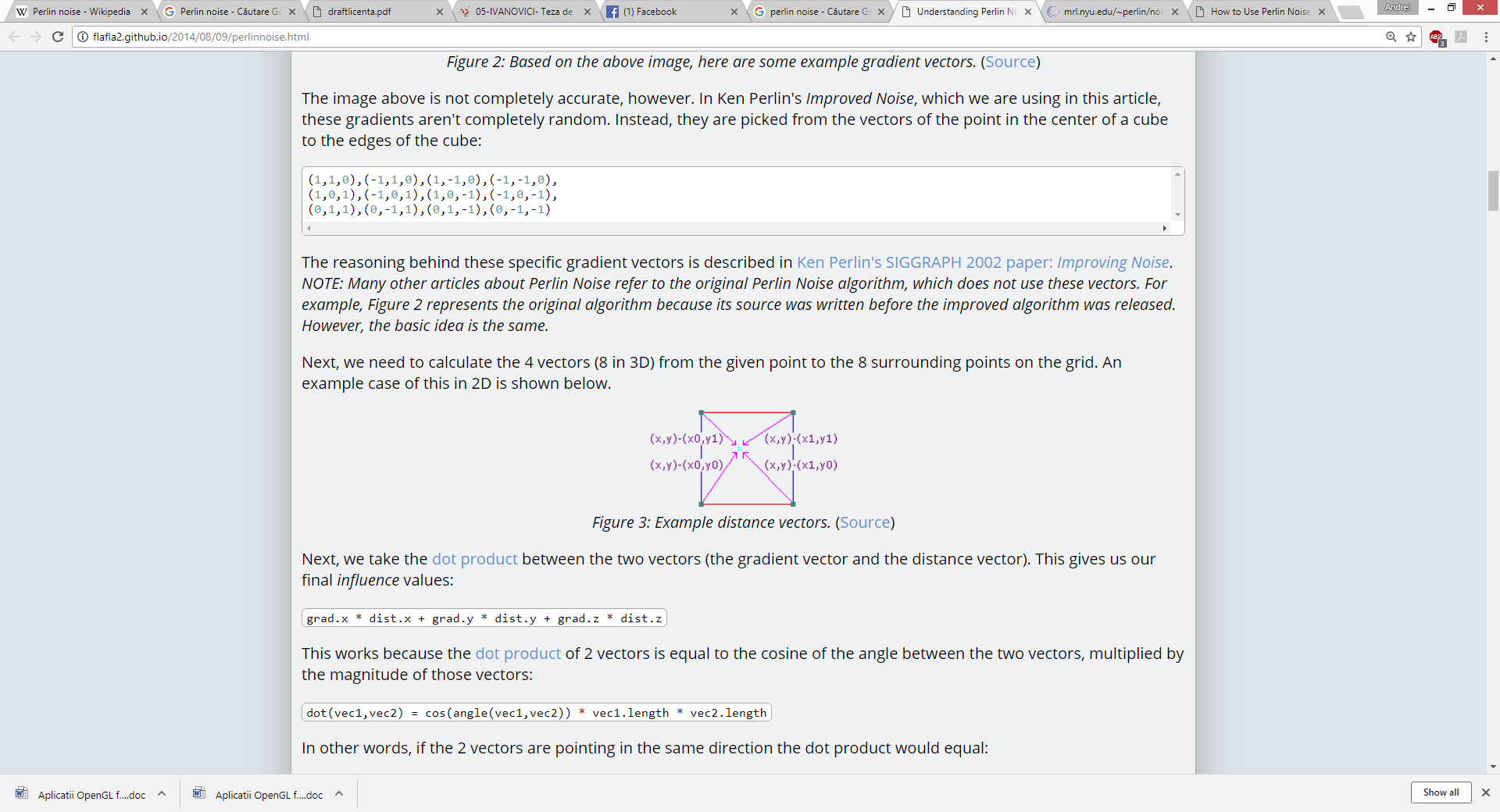
Unde x, y și z sunt numere reale.

Funcția va fi definită pe un spațiu în care pentru fiecare valoare întreagă există o dreaptă paralelă cu axa corespunzătoare într­un sistem cartezian, iar fiecare punct rațional se va afla într­un pătrat format din aceste drepte, după cum se poate observa în desenul următor:

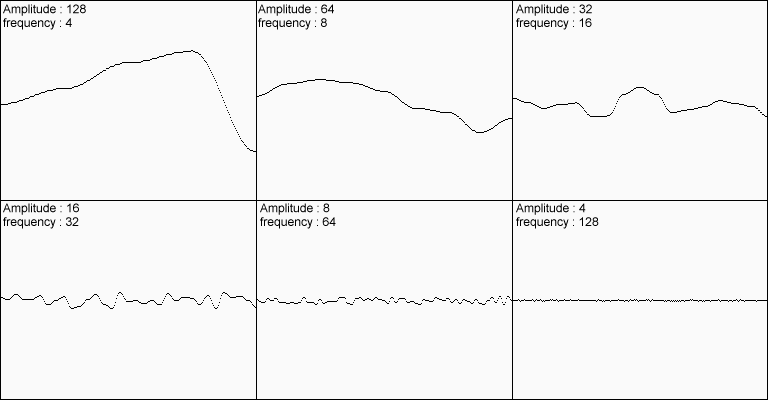
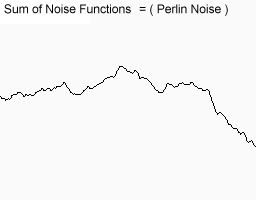
Următorul pas este definirea unei funcții care pentru fiecare punct din “grid” asignează un vector pseudo­random de lungime 1 în R​2, unde proprietatea “pseudo­random” înseamnă că vectorii sunt aparent aleatori, dar pentru același punct va return mereu același vector, iar fiecare vector are o șansă egală de a fi ales.

De asemenea, se va genera câte un vector și din fiecare punct al pătratului înspre punctul (x, y),după cum urmează:

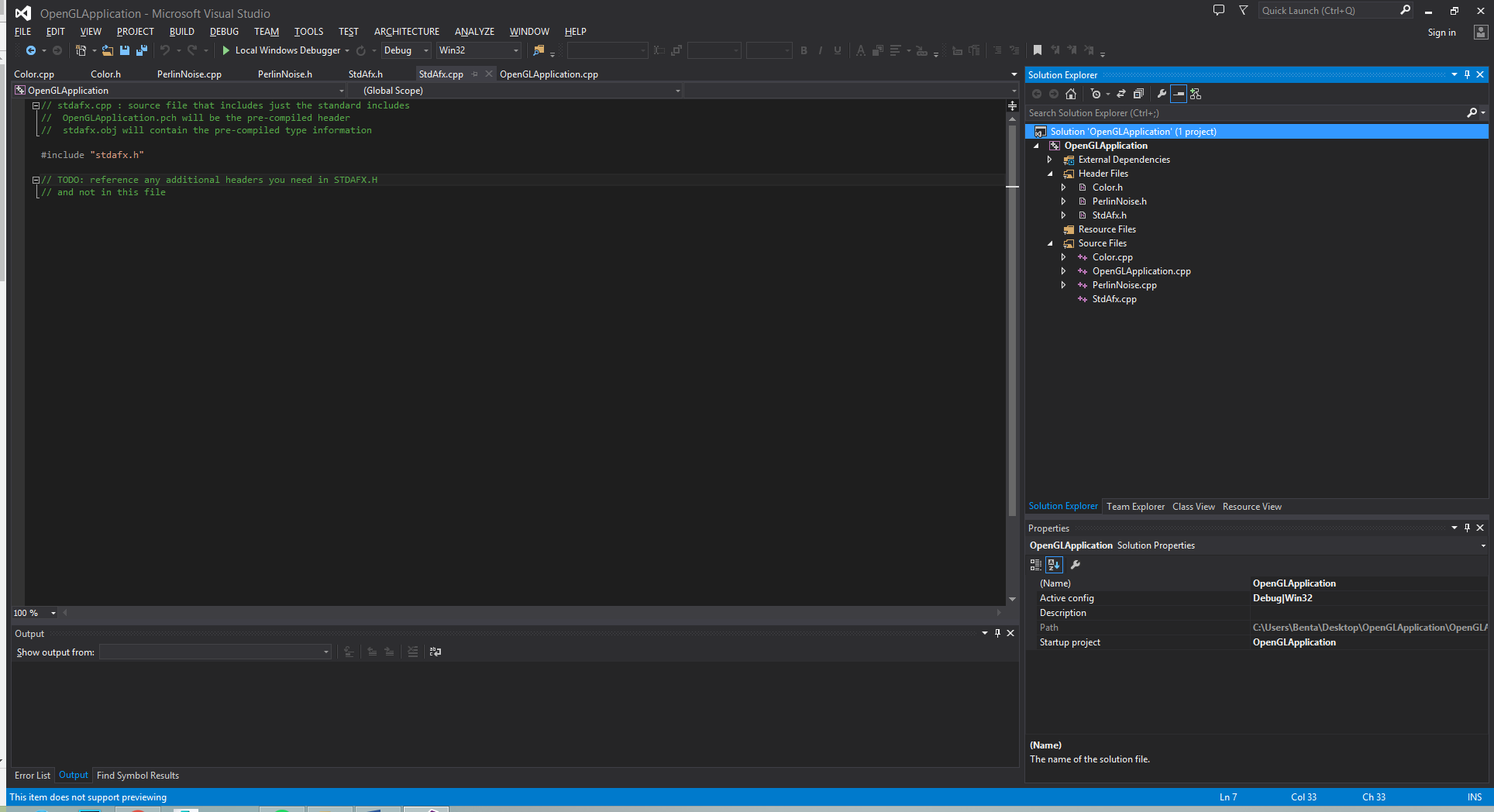
Calculul valorii efective a zgomotului în punctul (x, y) se face calculând influența fiecărui vector pseudo­random în output­ul final, acesta fiind o medie ponderată a acestor influențe. Reprezentarea grafică a unor valori s, t, u, v (acestea sunt pseudo­random!) arată în felul următor:

Valoarea zgomotului reprezintă o medie a acestor vectori de influență.

Pentru a avea rezultate realiste se folosește lucrul cu octave ceea ce presupune însumarea mai multor funcții de noise cu parametri diferiți. Acești parametri sunt amplitudinea și frecvența funcției. Pentru a determina cât de mult variază terenul utilizăm o valoare numită persistență și folosim următoarele formule: **frecventa = 2i** și **amplitudinea = persistențai** unde i este numărul de octave.

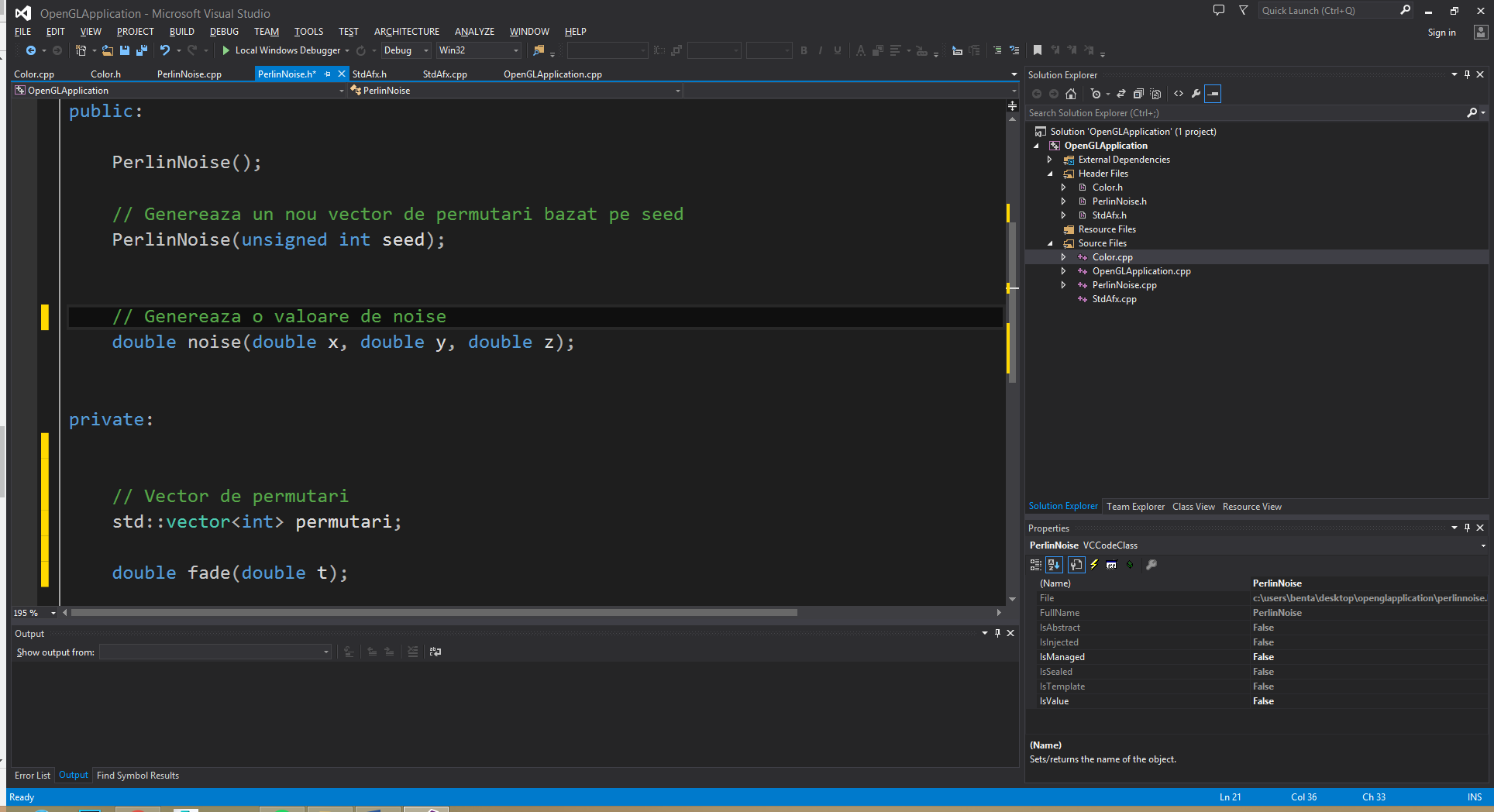
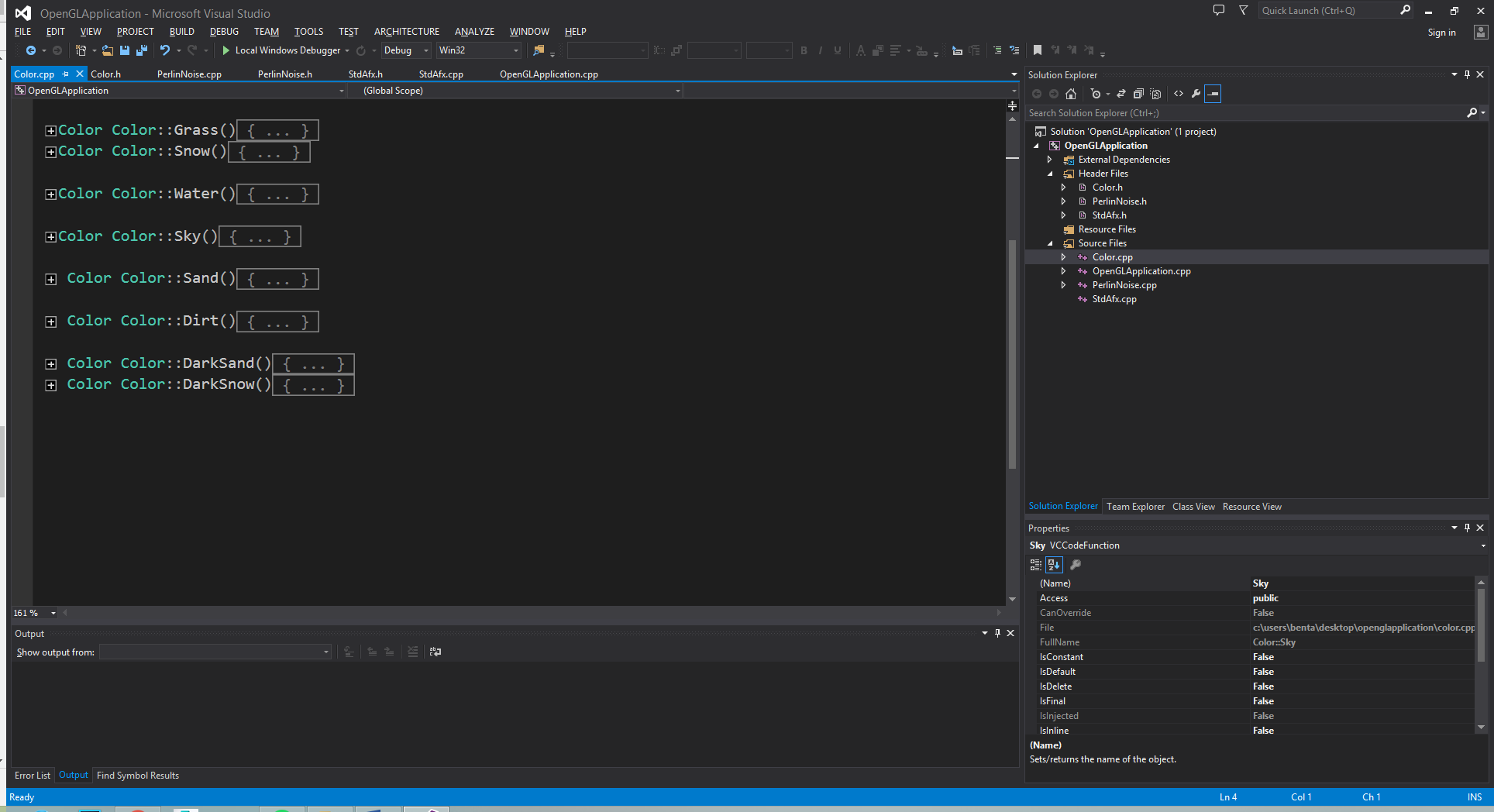


**Implementarea proiectului**

Proiectul e compus din fişierul sursă principal OpenGLApplication.cpp şi 2 clase, una numită Color care este folosită pentru a reţine date despre componentele roşu, verde şi albastru ale unei culori şi una PerlinNoise care implementează algoritmul de Perlin Noise prin declararea unui obiect de acest tip.

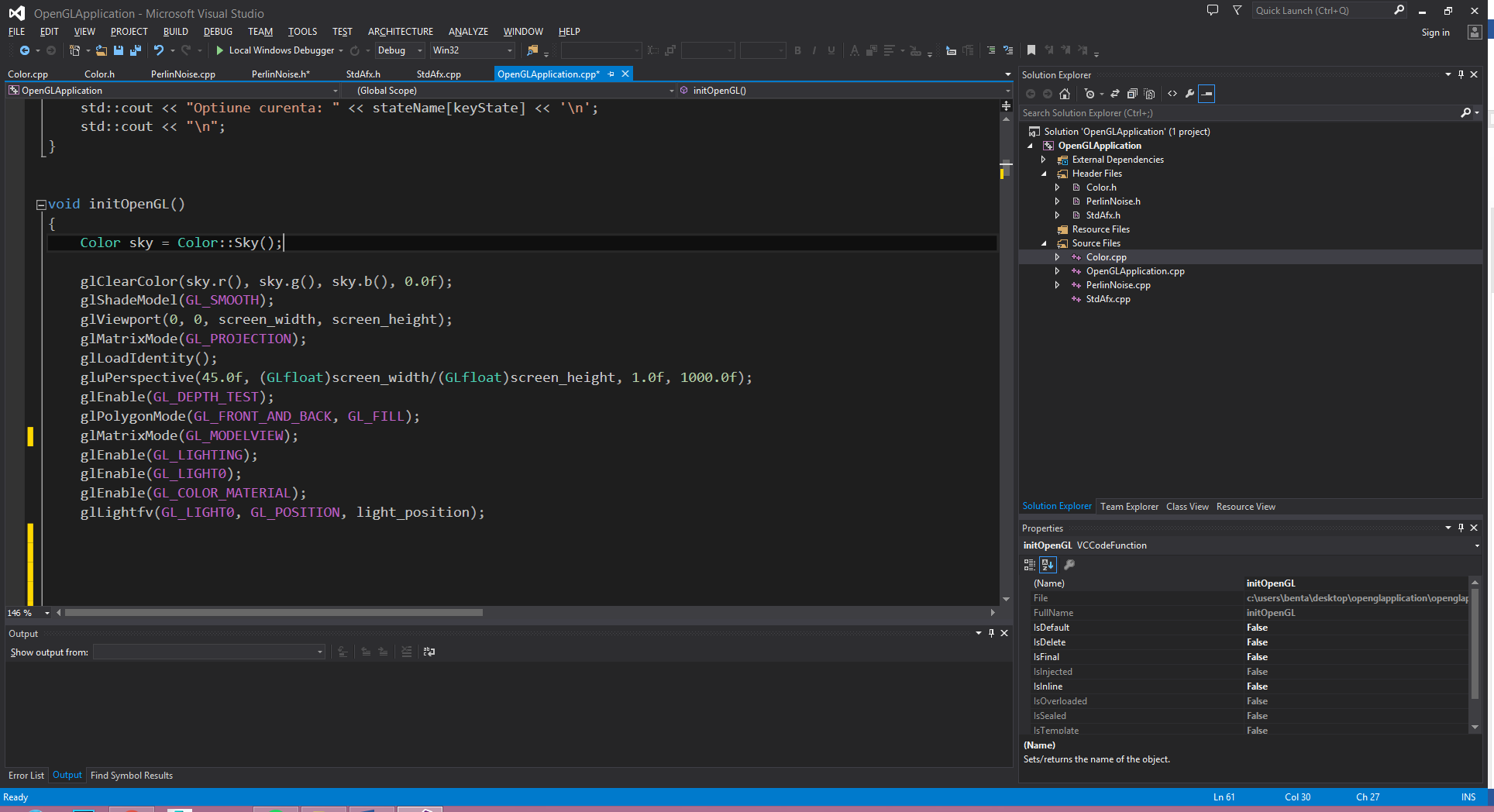
Clasa Color are ca date membre r, g, b de tip real cu valori cuprinse între 0 şi 1 ce reţin componentele roşu, verde şi albastru ale unei culori. Constructorul permite delcararea de obiecte prin specificarea acestor componente. De asemenea, clasa implementează funcţii statice ce returnează culori specifice precum culoarea ierbii, a pământului, a nisipului, etc.

Clasa PerlinNoise are ca date membre vectorul de permutari ce reţine date de tip întreg. De asemenea, clasa are şi funcţii membre necesare pentru aplicarea algoritmului precum funcţia de interpolare sau cea ce generează efectiv “noise map-ul” numită noise(). Constructorii clasei permit declararea de obiecte atât precizând seedul cât şi neprecizându-l, caz în care acesta este generat aleatoriu.



Cel mai relevant fișier este OpenGLApplication.cpp deoarece el găzduiește nucleul aplicației și în interiorul acestui fișier avem funcția **main()**. Funcția principală, impreună cu alte funcții specifice GLUT fac posibilă rularea aplicației și afișarea pe ecran a graficii 3D. În cele ce urmează vom trece în revistă câteva bucăți din nucleul aplicației.

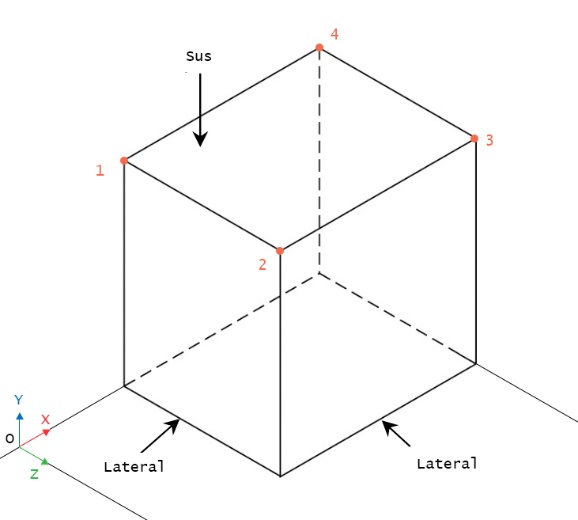
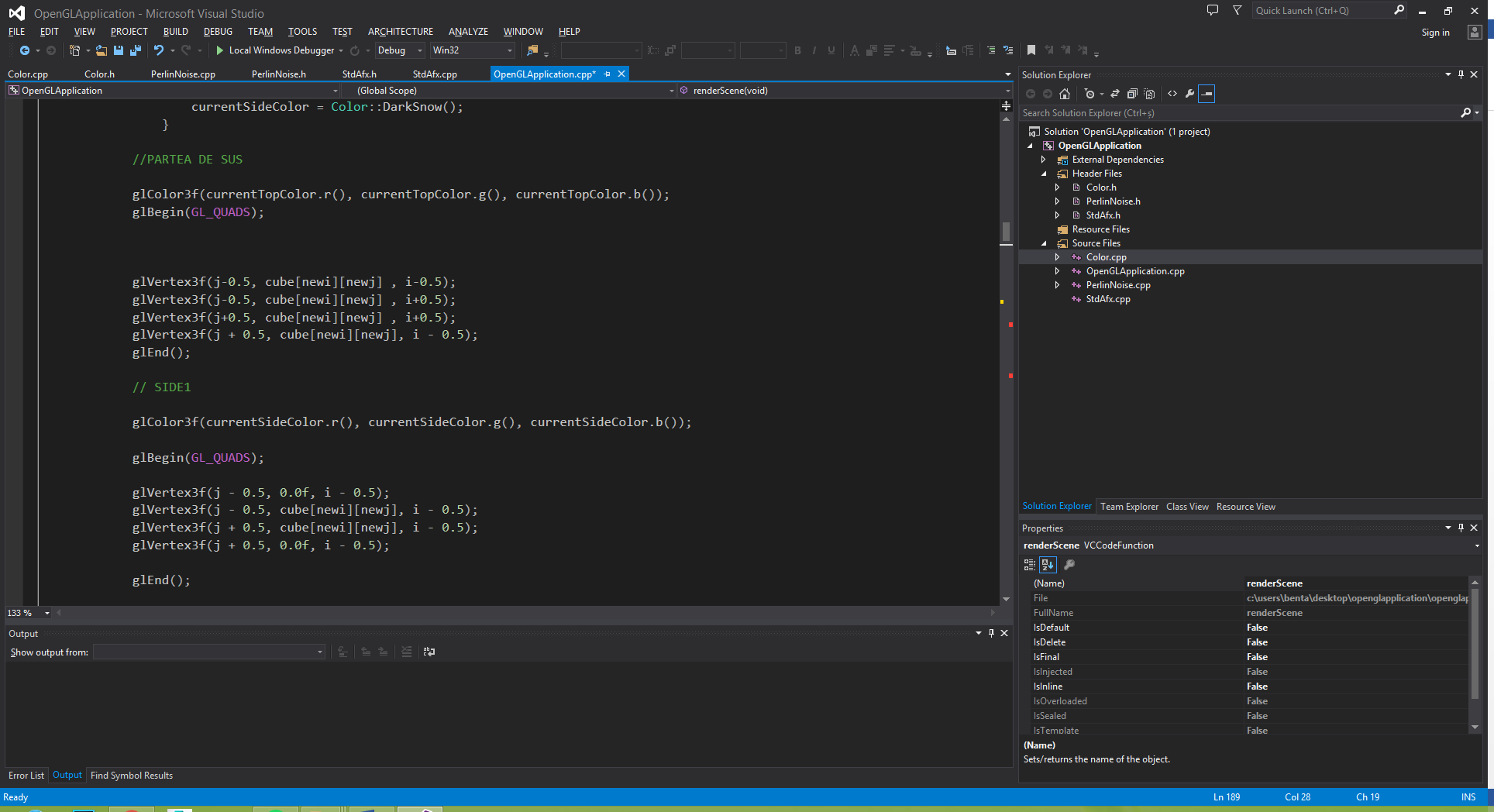
În **main()** se citesc de la tastatură dimensiunile planului (în număr de pătrate de unitate 1) pe care urmează să fie aplicat alforitmul. Pe lângă asta, funcția **main()** conține mai multe apeluri de funcții precum **gluInit()** (folosită pentru a inițializa biblioteca GLUT), **glutInitDisplayMode(),** **glutInitWindowPosition() glutInitWindowSize()** (folosite pentru a specifica date despre fereastră precum profunzimea de culoare, dimensiunea etc.), **glutDisplayFunc(), glutIdleFunc(), glutReshapeFunc(), glutKeyboardFunc()** (folosite pentru a specifica ce funcții se apelează atunci când GLUT desenează, fereastra se modifică, sunt apăsate taste etc.) etc. Tot aici se declară și obiectul de tip PerlinNoise. Către finalul funcției **main()** sunt două apeluri importante: **initOpenGL()** și **glutMainLoop()** care sunt folosite pentru a specifica amănunte în ceea ce privește inițializarea GLUT și respectiv pentru a intra in bucla principală.

 În funcția **initOpenGL()** se precizează culoarea de fundal, care implicit e o nuanță de albastru, se specifică tipurile de proiecție, câmpul de vizualizare, tipurile de iluminare și materiale etc.

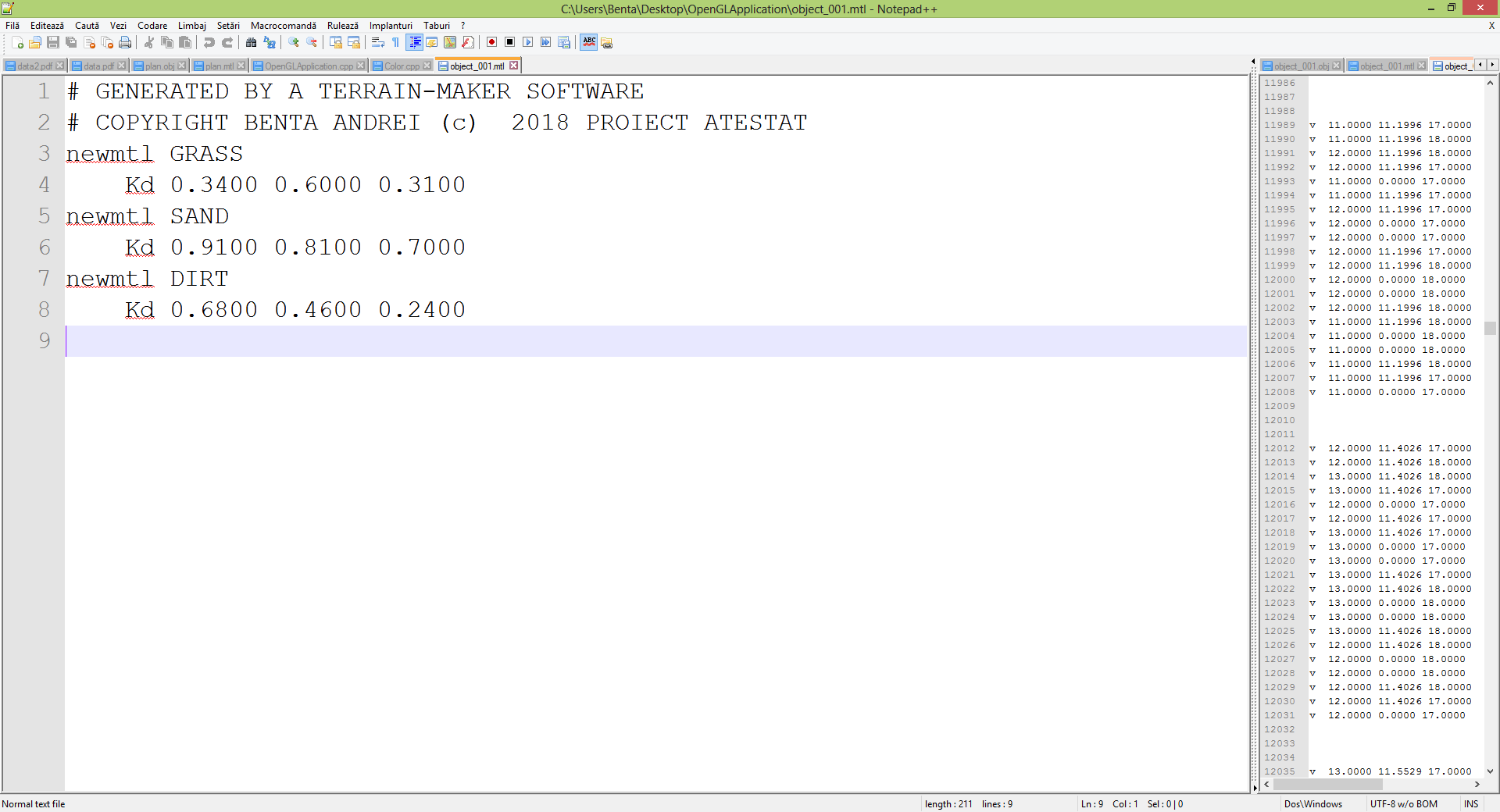
Funcția **processNormalKeys()** este una importantă deoarece fără aceasta nu am putea procesa tastele în timp real ceea ce nu i-ar oferi aplicației utilitate. Această funcție conține un **switch()** în care se procesează toatele taste necesare. Astfel tastele ‘i’,’j’,’k’,’l’,’y’,’h’ mută camera in scenă, tasta ‘z’ aplică Perlin Noise pe octave, tasta ‘s’ salvează obiectul 3D ,tasta ‘0’ schimbă opțiunea ce modifică parametrii funcției, iar tastele ‘+’ și ‘–‘ sunt folosite pentru a modifica parametrii funcției de Perlin Noise.

Funcţia **renderScene()** este apelată de fiecare dată când aplicaţia este inactivă sau procesează taste, la un interval de timp dat (framerate). Are o mare importanţă deoarece fără această funcţie nu ar fi posibilă partea interactivă vizuală. În cele ce urmează o sa prezint câteva detalii despre această funcţie. Prima metodă apelată este **glClear()** care curăţă ecranul de poligoane desenate în cadrul anterior, punând culoarea de fundal culoarea stabilită în **initOpenGL()** şi anume o nuanţă de albastru. Metoda **glLoadIdentity()** înlocuieşte matricea de transformări din program cu matricea unitate. Funcţia **gluLookAt()** este folosită pentru a-i spune “camerei” poziţia şi în ce direcţie să se uite şi este controlată de parametri care se schimbă prin apăsarea de taste.

Urmează desenarea planului ce reprezintă apa şi aceasta se realizează în felul următor: Întâi se specifică culoarea ce urmează să fie folosită cu ajutorul funcţiei **glColor3f()** (parametrii sunt r,g,b cu valori reale între 0 şi 1), apoi se apelează funcţia **glBegin()** (care are ca parametru tipul poligonului, în cazul de faţă este „quad” care înseamnă patrulater) care îi spune sistemului că urmează a fi precizate puncte pentru a desena poligonul, se precizează punctele prin apelul de patru ori a funcţiei **glVertex3f()** (care are ca parametri x,z,y valori reale), iar în final se apelează funcţia **glEnd()** care îi spune sistemului că a terminat de desenat poligonul. Este importantă ordinea precizării punctelor deoarece aceasta definește normala planului. Noi dorim ca vectorul normală să fie perpendicular pe plan și orientat “în sus” pentru ca planul să fie vizibil așa că vom preciza punctele în sens trigonometric.

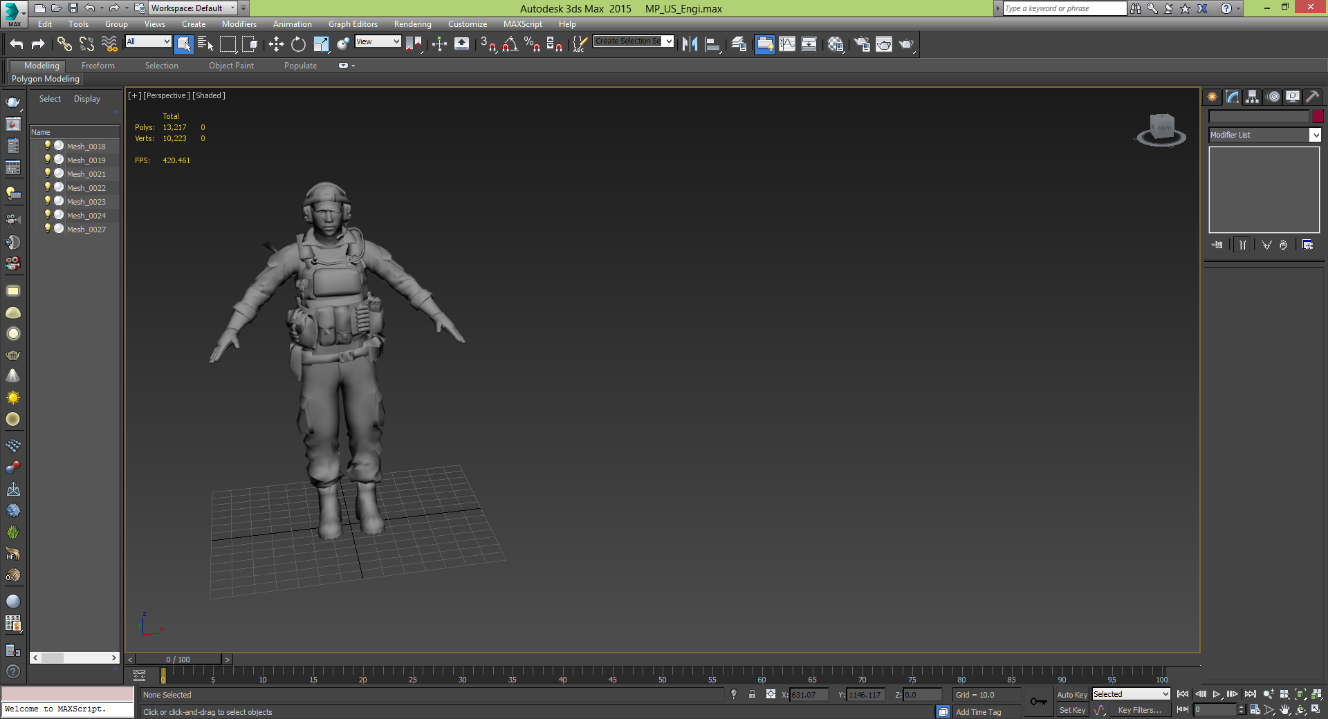
Următoarea secţiune este desenarea efectivă a cuburilor ce compun terenul. Întâi se specifică nuanţele folosite pentru a desena planele în funcţie de altitudine, iar apoi se realizează parcurgerea matricei de Perlin Noise, iar pentru fiecare valoare din matrice se desenează 4 plane perpendiculare pe pe planul format din mulţimea punctelor de pe axele X şi Z şi unul paralel, de lungimi ce depind de indicii şi respectiv valorile elementelor din matrice. Desenarea planelor se face punct cu punct. Se specifică vârfurile în sens trigonometric pentru cele 5 plane pentru a avea vectorul normală orientat corect. Avem o mică ilustrație în desenul următor:

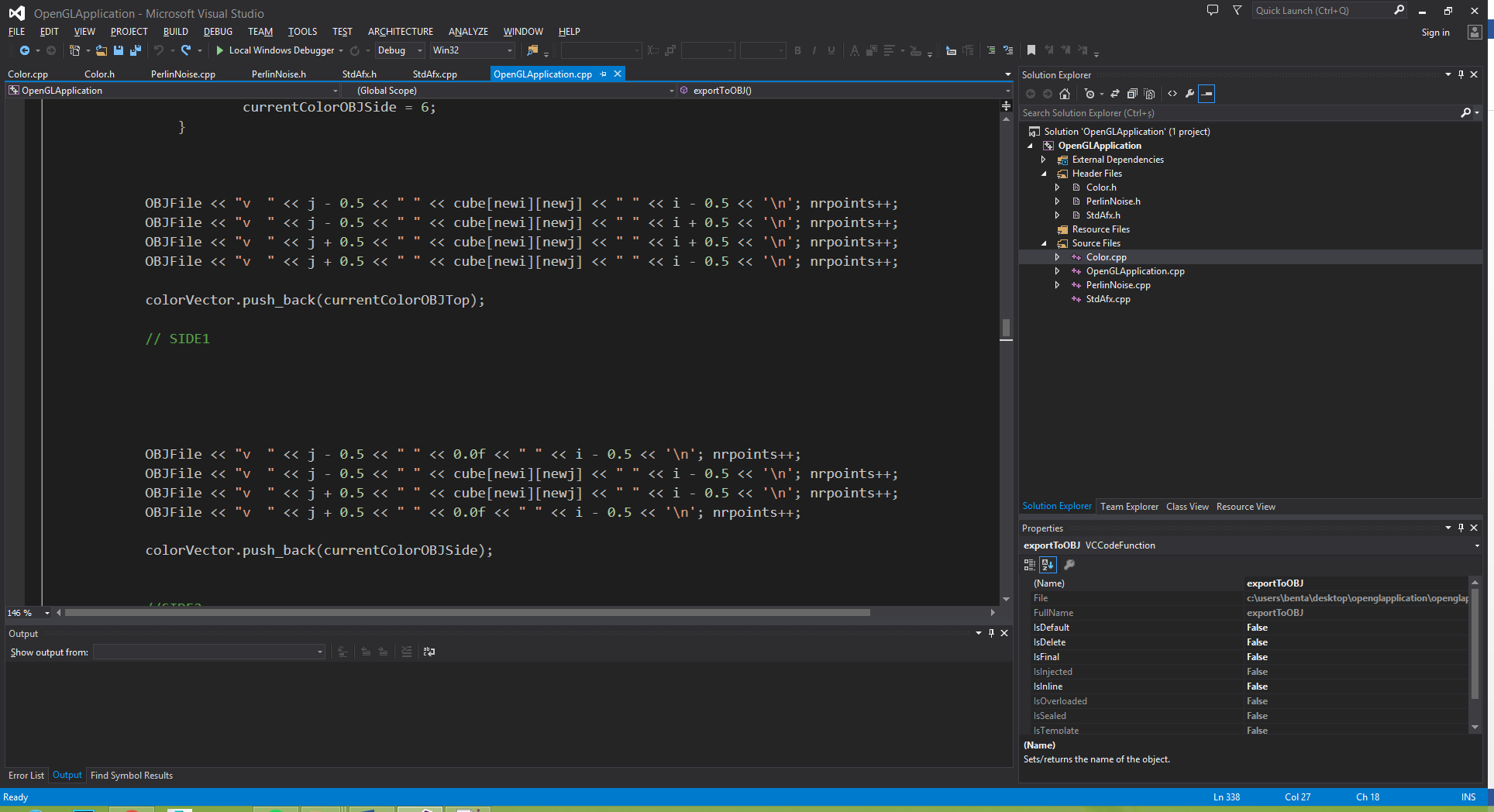
Funcția **aplicarePerlin()** aplică algoritmul și atribuie elementelor matricei cube[x][y] valorile generate. Funcția **octavePerlin()** aplică algortimul de Perlin Noise pe octave având un rezultat ceva mai realist. Funcțiile sunt separate deoarece **aplicarePerlin()** este apelată în timp real la schimbarea oricărui parametru și nu necesită timp mare de procesare față de **octavePerlin()** care e apelată la apăsarea unui buton de la tastatură deoarece timpul de executare este mai mare.

Funcția **exportToOBJ()** exportă datele într-un fisier de tip Wavefront .obj. Mecanismul e simplu și este realizat direct cu clasa std::ofstream. Se declară întâi un obiect de tipul acestei clase cu numele implicit “object\_001” și extensia .mtl ce va servi la specificarea culorilor din obiectul 3D. Formatul .mtl este asemănător cu .obj și se folosesc macrouri precum newmtl pentru a specifica un nou material sau Ka, Kd pentru a specifica culoarea folosită la ambient, diffuse etc.

După terminarea fișierului .mtl se declara un alt obiect de tip std::ofstream cu numele implicit „object\_001” și extensia .obj. În acesta se exportă datele despre coordonatele vârfurilor. Structura fișierului a fost explicată în capitolul anterior.

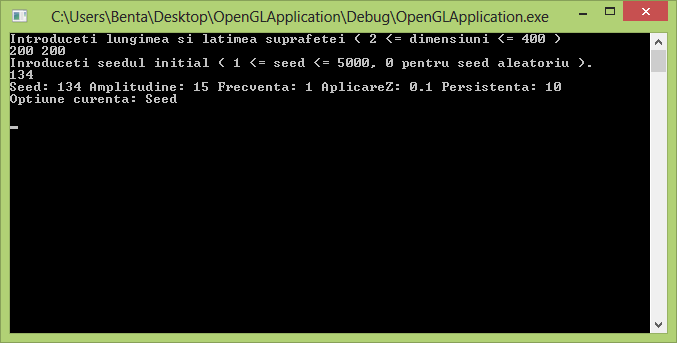
**Indicații de utilizare**

Întâi o sa vorbesc despre limitările tehnice ale programului. Dacă facem calculele matematice vom ajunge la concluzia că pentru o suprafață de lucru de 100x100 avem 10.000 de prisme patrulatere, 80.000 de vârfuri și 50.000 fețe (partea de jos nu se afișează), fiind echivalentul a mai multor soldați precum cel din imagine. Pentru a putea trasa toate aceste poligoane este necesară o placă video destul de modernă, de aceea alegerea unor dimensiuni mari va cauza probleme în funcționarea programului pe un calculator mai slab.



Cerințe recomandate:

* Procesor dual core modern (Intel Core I3) sau mai bun
* 2 GB RAM
* Placa video moderna cu 1024 MB (Nvidia Geforce GTS 450 ) sau mai bună



La lansarea programului prima fereastră este una de tip cmd care cere introducerea lungimii și a lățimii planului de lucru, dimensiuni cuprinse între 2 și 400, apoi necesită introducerea seedului. Seedul poate fi cuprins între 0 și 5000, iar în cazul în care se introduce 0, seedul va fi unul aleatoriu.

După ce aceste date au fost introduse, se va deschide fereastra propriu zisă cu terenul generat. Întâi dați click pe fereastra grafică pentru a face focus pe ea. În fereastra cmd se vor afișa parametrii și valorile lor și opțiunea curentă ce controlează un anume parametru.

De pe tastele ‘i’, ‘j’, ‘k’, ‘l’, ‘y’, ‘h’ navigați stânga , dreapta, sus, jos și respectiv zoom-in, zoom-out, de pe tasta ‘0’ schimbați opțiunea curentă, de pe tastele ‘+’ și ‘–‘ incrementați respectiv decrementați valoarea parametrului curent, iar de pe tasta ‘z’ aplicați Perlin Noise pe octave.