LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

“OpenGL datu vizualizācijas sistēma”

Kvalifikācijas darbs

Autors:
Kristaps Veitners
Studenta apl. Nr. Kv17044
Vadītājs:
Jānis Zuters
Dr.Dat.

RĪGA 2019

Anotācija

Kvalifikācijas darbs tika izveidots ar mērķi apgūt OpenGL grafiskās bibliotēkas pamatus, un saprast zinātnisko datu modelēšanu un vizualizēšanu, izmantojot ģenerēšanas, aprēķināšanas un matemātiskās tuvināšanas algoritmus. Lai realizētu šos mērķus tika izveidota OpenGL datu vizualizācijas sistēma, jo, lai efektīvi modelētu un vizualizētu datus ir nepieciešama augsta izpratne par matemātiskiem algoritmiem, OpenGL un 3D grafisko vidi. Kvalifikācijas darba gaitā ir izveidota programma ar 2D datu vizualizēšanu 3D vidē, iegūtas jaunas zināšanas par grafikas programmēšanu un gūts priekšstats par zinātnisku datu reprezentāciju.

Abstract

This project was designed with the goal of learning the basics of OpenGL graphics library and to further the understanding of scientific data modeling, visualizing, with the use of generation, calculation and mathematical approximation algorithms. To realise these goals an OpenGL data visualization system was created, because, to effectively model and visualize data it is required to have an in-depth understanding about mathematical algorithms, OpenGL and 3D graphical environment. During this project a program was created with 2D data visualization in a 3D environment, and new knowledge was gained about graphics programming and scientific data representation.

Atslēgvārdi

- OpenGL
- 3D
- Dati
- Vizualizācija
- Laukums

Saturs

1.Ievads

Izveidotā programmatūra realizē 2D datu vizualizēšanu 3D vidē, izmantojot laukuma ģenerēšanas, distances no centra un lineārās interpolācijas algoritmus. Vizualizējot datus, īpaši 3D kontekstā, ir nepieciešami ļoti lieli datu masīvi attiecībā pret ievadītajiem datiem, un ir jāatrisina problēma saistībā ar nezināmiem punktiem, kuriem ir jāpielieto matemātiskās tuvināšanas algoritmi. Tiek realizēta arī brīva kameras kustība, lai atļautu apskatīt ģenerēto datu modeli no visiem leņķiem.

Akurātai datu reprezentācijai 3D grafiskajā vidē ir arī jāpielieto normalizācijas funkcijas, jo ir iespējami datu zudumi ceļā no datu ielasīšanas līdz zīmēšanai uz ekrāna. Darba gaitā tika izpētītas arī dažādas lielu datu vizualizēšanas metodes, kā arī lielu bilžu ģenerēšana.

Lasot zinātniskos rakstus ir novērots, ka datu vizualizācijai ļoti bieži tiek izmantoti parasti 2D grafiki ar paskaidrojošo tekstu, parasti "Gnuplot", kas pats par sevi nav vizuāli iespaidīgs risinājums. OpenGL atļauj tiešu piekļuvi video kartes draiveru funkcijām, kas atļauj izveidot vizuāli iespaidīgas scēnas, izmantojot video spēļu grafisko dziņu izstrādāšanas metodes, jo mūsdienu video spēles ir ļoti vizuāli iespaidīgas.

Projekta valoda tika izvēlēta C++, jo tai ir liela saderība ar C valodu, kurā ir sarakstītas oriģinālās OpenGL bibliotēkas. C++ arī atļauj viegli pārvaldīt izmantotos resursus.

2.Pamatdefinīcijas un saīsinājumi

- Virsotne – Datu kolekcija, kas reprezentē punktu.
- GPU – Grafiskais procesors.
- Virsotnes buffera objekts – Punktu koordinātu datu kolekcijas, kas tiek saglabātas grafiskā procesora atmiņā.
- Virsotnes masīva objekts – Glabā virsotnes bufferā esošo datu stāvokļa informāciju.
- Normalizētas ierīču koordinātes – Koordinātu sistēma pēc normalizācijas, kurā visi virsotņu punkti atrodas robežās no -1.0 līdz 1.0, pirms attēlošanas uz ekrāna.
- Virsotņu/Fragmentu shaderis – GPU programma, kas balstoties uz padotajiem datiem izrēķina punktu pozīcijas un krāsas.
- Shadera programma – Virsotņu un fragmentu shaderu apvienojums vienā programmā.
- GLSL – Grafiskās bibliotēkas shaderu valoda.
- Uniforma – Globāls GLSL mainīgais, kam var piekļūt jebkurš shaderis shadera programmā.
- Elementu buffera objekts – bufferis indeksu glabāšanai priekš indeksētas grafiskās zīmēšanas.
- Poligons – mazākais daudzstūris, ko var izvadīt 3D vidē.
- Matrica – vektoru skaitļu masīvs, kurš tiek izmantots scēnas grozīšanai/manipulēšanai.

3.Lietotājstāsti

Programmatūras izstrādei tika izmantota spējā pieeja. Jauni uzdevumi un prasības tika izvirzītas balstoties un iepriešējo uzdevumu veiksmīgu izpildīšanu, lai nepavadītu pārāk daudz laika pie neefektīviem risinājumiem.

Lietotājstāsti ir sadalīti divos lielos posmos, no kuriem katrs ir par savu izstrādes virzienu. Katrs posms ir sadalīts iterācijās, kur tiek realizēta sava funkcionalitāte.

3.1 Posms 1 – vizualizēt 2D datus 3D vidē.

3.1.1 Iterācija 1

3.1.1. tabula

Lietotājstāsts/uzdevums	Sīkāks apraksts
Uz ekrāna ir jāredz objekts.	• Savienotas OpenGL bibliotēkas. • Izveidots OpenGL grafiskais logs. • Izveidota OpenGL shader klase. • Uzrakstīti primitīvi shaderi. • Uz ekrāna izvadīts grafisks objekts.

3.1.2 Iterācija 2

3.1.2 tabula

Lietotājstāsts/uzdevums	Sīkāks apraksts
Uz ekrāna ir jāredz funkcija.	• Uzrakstīts algoritms funkcijas aprēķināšanai. • Uzrakstīts algoritms funkcijas normalizēšanai. • Uz ekrāna grafiski izvadīta funkcija.
Funkcijai ir jākustās.	• Izmantotas Uniformas laika soļa iegūšanai un reāllaika funkcijas manipulēšanai.

3.1.3 Iterācija 3

3.1.3 tabula

Lietotājstāsts/uzdevums	Sīkāks apraksts
Uz ekrāna ir jāredz vairāku poligonu objekts.	• Izveidots elementu buffera objekts. • Uz ekrāna izvadīts divu poligonu grafisks objekts izmantojot elementu buffera objektu.
Objektam ir jābūt 3D vidē.	• Izveidota perspektīva, skata un modeļa matrica lai realizētu 3D vidi.

3.1.4 Iterācija 4

3.1.4 tabula

Lietotājstāsts/uzdevums	Sīkāks apraksts
Uz ekrāna ir jāredz 100x100 punktu laukums.	• Uzrakstīts algoritms, kas izveido 100x100 punktu laukumu un sastāv no 20000 poligoniem. • Pamainīta perspektīva, skata un modeļa matrica lai labāk redzētu laukumu.
Jārealizē 3D funkcijas attēlošana.	• Uzrakstīts algoritms distances aprēķināšanai no centra. • Modificēts iepriekš uzrakstītais algoritms funkcijas aprēķināšanai, lai tas pieņemtu vērtības balstoties uz distanci no centra.
Funkcijai jābūt kustīgai.	• Uz ekrāna izvadīta kustīga 3D funkcija izmantojot uniformas.

3.2 Posms 2 – pielikt klāt papildus funkcionalitāti.

3.2.1 Iterācija 1

3.2.1 tabula

Lietotājstāsts/uzdevums	Sīkāks apraksts
Funkcijas laukumam jābūt savādākā krāsā attiecībā pret funkcijas vērtībām.	• Uzrakstīts shaderis funkcijas intensitātes krāsai.
Jāvar aizvērt logu ar (ESC) taustiņu.	• Pielikta papildus funkcija taustiņu nolasīšanai.
Funkcijas laukumam jāmaina savi izmēri attiecībā pret programmas loga lielumu	• Pielikta funkcija loga palielināšanai/samazināšanai.

Lai izmainītu kaut ko programmā ir pārāk daudz vietās jāizmaina vieni un tie paši skaitļi.	• Veikta programmas optimizācija, vairāki literāļi nomainīti ar mainīgo.

3.2.2 Iterācija 2

3.2.2 tabula

Lietotājstāsts/uzdevums	Sīkāks apraksts
Programmai ir jāstrādā ar ārēji ievadītiem datiem.	• Realizēta datu nolasīšana no faila.
Uz ekrāna ir jāredz 2D dati 3D vidē.	• Realizēts lineārās interpolācijas algoritms un dati vizualizēti uz punktu laukuma.

3.2.3 Iterācija 3

3.2.3 tabula

Lietotājstāsts/uzdevums	Sīkāks apraksts
Uz ekrāna atrodas tikai poligonu laukums.	• Pieliktas asis.
Caurspīdīgu poligonu laukumu ir grūti saprast, kopā ar asu līnijām.	• Renderēts vēlviens objekts, tikai ar līnijām, labākai reprezentācijai.

3.2.4 Iterācija 4

3.2.4 tabula

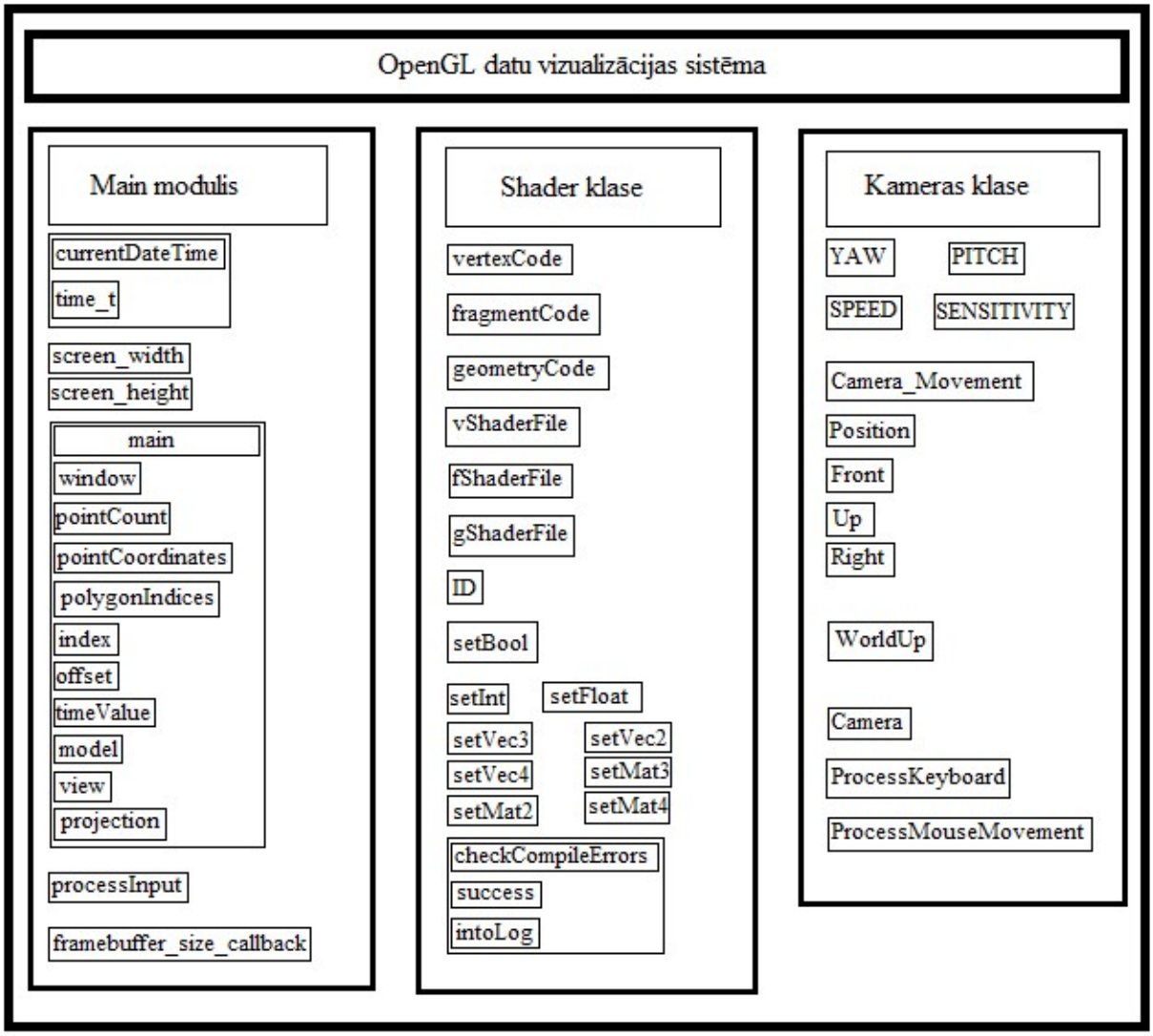
Lietotājstāsts/uzdevums	Sīkāks apraksts
Jābūt iespējai apskatīt 3D laukumu no visiem iespējamiem leņķiem.	• Izveidota brīvas kustības kameras klase.

4.Programmas projektējuma apraksts

4.1 Ievads

Projekts sastāv no trīs moduļiem, “Shaderu klase”, “Kameras klase” un “Main funkcija”. “Shaderu klase” un “Kameras klase” ir universālas datu struktūras, tās var pievienot un lietot jebkurā citā OpenGL programmā, kas izmanto atbilstošu OpenGL versiju. Modulis “Main funkcija” sevī ietver visu projektu un izmanto “Shaderu klasi” un “Kameras klasi”. Tā galvenā funkcija ir main(), kas izveido OpenGL logu, nolasa informāciju no faila un izpilda nepieciešamos algoritmus. Tā galvenā sastāvdaļa ir windowshouldclose() funkcija, kas atļauj programmai turpināt zīmēt uz ekrāna un izmantot reāllaika manipulācijas līdz tiek izsaukta programmas aizvēršana.

Tā kā projekta galvenā daļa ir izmantotie algoritmi, programmas izstrādes gaitā tika izveidoti tikai divi objekti.



Attēls 4.1: Projekta moduļi, objekti, funkcijas un galvenie mainīgie

4.2 Shader klases apraksts

Shader klase ir izveidota ar mērķi palielināt programmas abstrakciju. Galvenā funkcija ir GLSL failu ielasīšana no sistēmas, to satura ierakstīšana mainīgajos, failu atbrīvošana un shadera programmas izveide. Ar šo klasi ir iespējams izveidot vairākas shadera programmas viena projekta lietošanai, kas arī tiek realizēts projektā.

4.2.1 Mainīgie

- Const char* vertexPath – ceļš uz virsotņu shadera failu.
- Const char* fragmentPath – ceļš uz fragmentu shadera failu.
- Std::string vertexCode – string mainīgais virsotņu GLSL koda turēšanai.
- Std::string fragmentCode – string mainīgais fragmentu GLSL koda turēšanai.
- Std::ifstream vShaderFile – mainīgais GLSL faila turēšanai.
- Std::ifstream fShaderFile – mainīgais GLSL faila turēšanai.
- Unsigned int vertex –virsotņu shadera ID.
- Unsigned int fragment – fragmentu shadera ID.
- Unsigned int ID – shadera programmas mainīgais.
- Glint success – glabā informāciju vai shaderu kompilēšana notikusi veiksmīgi.
- Glchar infoLog – glabā informāciju par erroriem, kas radušies shaderu kompilācijas laikā.

4.2.2 Funkcijas

- Void use()
    - Aktivizē izveidoto shadera programmu ar glUseProgram(ID).
- Shader(const char* vertexPath, const char* fragmentPath)
    - Iegūst virsotņu un fragmentu pirmkodu no failiem.
    - Kompilē shaderus.
    - Izveido shaderu programmu.
    - Atbrīvojas no izmantotajiem resursiem.
- Void createBool()
    - Funkcija Bool uniformas izveidošanai.
- Void createInt()
    - Funkcija Int uniformas izveidošanai.
- Void createFloat()
    - Funkcija Float uniformas izveidošanai.
- Void createVec2()
    - Funkcija Vec2 uniformas izveidošanai.
- Void createVec3()
    - Funkcija Vec3 uniformas izveidošanai.
- Void createVec4()
    - Funkcija Vec4 uniformas izveidošanai.
- Void createMat2()
    - Funkcija Mat2 uniformas izveidošanai.
- Void createMat3()
    - Funkcija Mat3 uniformas izveidošanai.
- Void createMat4()
    - Funkcija Mat4 uniformas izveidošanai.
- Void checkCompileErrors(Gluint shader, std::string type)
    - Izvada kļūdu paziņojumus, ja shaderu kompilēšana nav bijusi veiksmīga.

4.3 Kameras klases apraksts

Kameras klase realizē kustību cauri scēnai, nolasot lietotāja klaviatūras taustiņu ievadu un attiecīgi izmainot scēnas skata, modeļa un projekcijas matricu. Tiek izmantotas arī trīs leņķa vērtības, kas kopā veido pagrieziena leņķa vektoru, kas kopā ar skata, modeļa un projekcijas matricu atļauj atrasties jebkurā vietā scēnā, jebkādā pagrieziena leņķī.

4.3.1 Mainīgie

- enum Camera_Movement – Satur četras dažādas iespējas kameras kustībai - uz priekšu, atpakaļ, pa kreisi, pa labi.
- Const float YAW – Sākuma stāvokļa leņķa pagrieziena vērtība.
- Const float PITCH - Sākuma stāvokļa leņķa pagrieziena vērtība.
- Const float SPEED – Kustības ātruma vērtība.
- Const float SENSITIVITY – Peles pagrieziena jūtības vērtība.
- Glm::vec3 Position - kameras pozīcijas vektors.
- Glm::vec3 Front – kameras x ass vērtība.
- Glm vec3 Up – kameras y ass vērtība.
- Glm vec3 Right – kameras z ass vērtība.
- Glm::vec3 WorldUp – papildus mainīgais, kas izmantots priekš pagrieziena normalizācijas funkcijas.
- Float Yaw – kameras rotācija ap x asi.
- Float Pitch – kameras rotācija ap y asi.
- Glm::vec3 front – atsevišķs mainīgais updateCameraVectors() funkcijā rotācijas realizācijai ap z asi.

4.3.2 Funkcijas

- Camera()
    - Kameras konstruktora funkcija, izmanto sākuma stāvokļā vērtības.
- Glm::mat4 GetViewMatrix()
    - Atgriež scēnas skata matricu ar jaunām vērtībām
- Void ProcessKeyboard()
    - Izmaina kopējo scēnu pretēji klaviatūras ievadītajam virzienam, lai veidotu ilūziju, ka lietotājs pats kustās pa 3D vidi.
- Void ProcessMouseMovement()
    - Balstoties uz peles jūtīgumu un kustības virzienu maina scēnas atrašanās leņķi pretēji peles kustības leņķim.
- Void updateCameraVectors()
    - Pēc katras programmas darbības laika vienības tiek pārrēķināta un pielietota jauna kameras pozīcija.

4.4 Main moduļa apraksts

Main modulis realizē visu projekta funkcionalitāti. Tas izveido OpenGL grafisko logu, izmanto shadera klasi, lai izveidotu atsevišķiem modeļiem nepieciešamos shaderus, ar algoritmiem vizualizē zinātniskos datus un izmanto kameras klasi, lai tos attēlotu no jebkāda skata leņķa.

4.4.1 Mainīgie

- Int screen_width – Sākuma ekrāna platums.
- Int screen_height – Sākuma ekrāna augstums.
- GLFWwindow* window – OpenGL loga objekts.
- Int pointcount – Punktu skaits laukuma ģenerēšanai.
- Float *pointCoordinates – koordinātu masīvs punktiem, katram punktam trīs koordinātas.
- Int *polygonIndices – poligonu indeksu masīvs, satur 99x99x2x3 vērtību(kvadrātu skaits, katrā kvadrātā divi poligoni un katram poligonam trīs koordinātas).
- Int index – izseko indeksiem indeksu aizpildīšanas algoritmā.
- Int offset – katrā nākošajā laukuma ģenerēšanas iterācijā jāpakustās vienu rindu uz priekšu.
- Shader myShader – laukuma shadera objekts.
- Shader anotherShader – asu shadera objekts.
- Unsigned int VBO – virsotņu buffera objekts.
- Unsigned int VAO – virsotņu masīva objekts.
- Unsigned int EBO – elementu bufera objekts.
- Float timeValue – reāllaika vērtības mainīgais.
- Glm::mat4 projection – projekcijas matricas uniforma, kas tiek padota shaderim.
- Glm::mat4 view – kameras skata transformācijas matrica, kas tiek padota shaderim.
- Std::ifstream dataFile – mainīgais ielasīto datu turēšanai.
- Float t – distance no centra.
- Float z – interpolēta vērtība.

4.4.2 Funkcijas

- Const std::string currentDateTime()
  - Funkcija laika un datuma iegūšanai priekš logošanas.
- Main()
  - Main funkcija izveido OpenGL logu.
  - Ģenerē laukumu.
  - Nolasa datus no faila.
  - Piesauc interpolācijas funkciju un aprēķina nezināmos punktus.
  - Izveido shaderus piesaucot shaderu klasi.
  - Konfigurē VBO, VAO, EBO grafiku zīmēšanai.
  - Sāk un uztur whilewindowshouldclose() ciklu.
    - Ciklā tiek zīmēta scēna un uzturēts kameras objekts.
- Void processInput()
  - Nolasa klaviatūras ievadu, aizver logu, ja nospiests (ESC).
- Void framebuffer_size_callback()
  - Liek scēnai mainīties logam palielinoties/samazinoties.
- Double interpolate()
  - Atgriež interpolētos datus.

4.5 Datu vizualizēšanas algoritmu apraksti

4.5.1 Ģenerēšanas algoritms

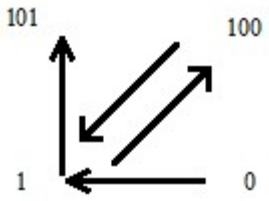
Tiek definēti divi mainīgie, viens punktu koordinātu (virsotņu) turēšanai un otrs indeksu turēšanai.

Punktu koordinātu mainīgais sastāv no 100x100x3=30000 elementiem, katram punktam sava xyz koordināte.

Indeksu mainīgais sastāv no 99x99x2x3= 58806 elementiem, 99x99 kvadrātu laukums, katrs kvadrāts sastāv no diviem trīsstūriem(poligoniem), un katrs poligons sastāv no trīs punktiem.

Lai aizpildītu koordinātu masīvu, tiek iets cauri 100x100 cikliem, kuros masīvam tiek piešķirta koordināte balstoties uz pašreizējo iterācijas vērtību un aizpildāmo mainīgā koordinātu asi.
Masīva indekss tiek aprēķināts ar formulu (i * pointCount + j).
Punkta koordinātes vērtība tiek piešķirta ar formulu (i / 10.0f). Kur i tiek nomainīts uz j gadījumā, ja ir jāaizpilda otra koordinātu ass vērtība.

Lai aizpildītu indeksu masīvu, tiek iets cauri 99x99 cikliem, kuros balstoties uz index un offset vērtībām tiek izveidoti divi poligoni ar 6 indeksu vērtībām, index tiek palielināts par 6 nākošajai iterācijai. Offset tiek aprēķināts no iterācijas vērtībām.



Attēls 4.5.1, indeksu aizpildīšanas cikla vērtību piešķiršanas secība

4.5.2 Distances no centra formula

$$t = \sqrt{x^2 + y^2}$$

Attēls 4.5.2, distances no centra formula

X un Y vērtības iegūst ejot cauri 100x100 cikliem un aprēķinot katram punktam atrašanās vietu attiecībā pret centru ar formulu (i – pointCount / 2) / (pointCount / 2.0f).

Pati formula tiek aprēķināta izmantojot iebūvētu C++ hypotf funkciju.

4.5.3 Lineārā interpolācija

$$y = y_1 + \frac{(y_2 - y_1)(x - x_1)}{(x_2 - x_1)}$$

Attēls 4.5.3, lineārās interpolācijas formula

Interpolācijas vērtības iegūst ejot cauri 100x100 cikliem un katrai distances no centra vērtībai piesaucot Interpolācijas funkciju.

Ar zināmiem ievaddatiem distance no centra tiek pietuvināta tuvākajam elementam, to salīdzinot ar zināmām datu vērtībām.

Kad tuvākais skaitlis ir sasniegts tiek aprēķināts gradients atņemot labo y vērtību no kreisās un izdalot ar tādu pašu x izteiksmi. Gradient = (rightY – leftY) / (rightX – leftX)

Ar iegūto Gradient skaitli var veikt lineāru interpolāciju.
InterpolatedValue = leftY + Gradient * (x – leftX). X šajā kontekstā ir koordināte kurai tiek veikta interpolācija y vērtībai.

5.Projekta organizācija

Darbu izstrādāju viens cilvēks, bez saziņas vai palīdzības no citiem.

Projekts sastāv no trīs koda failiem un vairākiem GLSL shadera failiem, priekš atsevišķiem modeļiem. Divi koda faili ir neatkarīgi moduļi, ko var izmantot citās programmās, un tie nodrošina kameras funkcijas un shaderu programmas izveidošanu. Trešais koda fails ir galvenais main modulis, kas sastāv no nepieciešamām funkcijām un izmanto visus pārējos failus un bibliotēkas grafikas apstrādei un datu vizualizācijai.

6.Konfigurāciju pārvaldība

Projektam netika lietotas nekādas versijošanai paredzētas programmas, jo grafiskus projektus ir grūti migrēt starp sistēmām nepieciešamo bibliotēku dēļ, kā arī darbu izstrādāja tikai viena persona.

Projekta versijošana sastāvēja no manuālas kopijas izveidošanas pēc katras samērā lielas funkcionalitātes ieviešanas. Projektam ir vairākas versijas, kur katrā ir realizēta liela mēroga funkcionalitāte.

7.Kvalitātes nodrošināšana

Kvalitāte tika nodrošināta testējot esošo funkcionalitāti un izveidojot logošanas un kļūdu ziņojumus, kas atļāva ātri un ērti atrast kļūdas pēc jaunas funkcionalitātes ieviešanas. Testēšanai nereti tika lietoti “Microsoft Visual Studio” atkļūdošanas rīki.

Paralēli testiem projekts arī tika salīdzināts ar citām, līdzīgām datu vizualizācijas programmām un no tām tika aizgūtas idejas augstākai funkcionalitātei.

8.Darbietilpības novērtējums

Pirms šī projekta izstrādāšanas autoram nebija liela pieredze 3D grafikas programmēšanā, zinātnisku datu modelēšanā un vizualizēšanā. Vajadzīgo algoritmu atrašana, saprašana un realizēšana programmā arī pagarināja posmu izpildes laiku. Visvairāk laika aizņēma 3D matemātikas saprašana un realizēšana, uz kā balstās 3D grafiskā vide.

Viens posms vidēji aizņēma pusotru mēnesi.

8.1 Pirmais posms.

Pirmā iterācija – 3 dienas.

Otrā iterācija – 4 dienas.

Trešā iterācija – 5 dienas.

Ceturtā iterācija – 15 dienas.

Kopā – 27 dienas.

8.2 Otrais posms.

Pirmā iterācija – 4 dienas.

Otrā iterācija – 10 dienas.

Trēšā iterācija – 5 dienas.

Ceturtā iterācija – 6 dienas.

Kopā – 25 dienas.

9.Testēšanas dokumentācija

Pēc katra iterācijas posma bija izveidots akcepttests, kuru pārbaudīja izstrādātājs. Vizualizācijas sistēmai testēšana īpašas grūtības nesagādāja, jo ieviestā logošanas funkcija ātri un vienkārši norādīja uz kļūdainām koda daļām, kā arī vizuāli bija viegli identificēt nepareizu darbību.

9.1 tabula

Lietotājstāsti	Akcepttests	Sagaidāmais rezultāts	Testa rezultāts
Uz ekrāna ir jāredz objekts.	1. Palaist programmu. 2. Attēlot objektu.	1. Programmas logs atveras. 2. Objekts ir redzams.	1. Veiksmīgs. 2. Veiksmīgs.
Uz ekrāna ir jāredz funkcija.	1. Palaist programmu ar funkcijas parametriem.	1. Funkcija ir redzama.	1. Veiksmīgs.
Funkcijai ir jākustās.	1. Palaist programmu ar Uniform reāllaika funkciju.	1. Funkcija kustās.	1. Veiksmīgs.
Uz ekrāna ir jāredz vairāku poligonu objekts.	1. Palaist programmu ar vairāku poligonu objektu.	1. Ir redzams divu poligonu četrstūris.	1. Veiksmīgs.
Objektam ir jābūt 3D vidē.	1. Palaist programmu ar izstrādātu 3D vidi.	1. Objekts izskatās it kā tas būtu uz grīdas.	1. Veiksmīgs.
Uz ekrāna ir jāredz 100x100 punktu laukums.	1. Palaist programmu ar 100x100 punktu modeli.	1. Ir redzams 100x100 punktu laukums.	1. Veiksmīgs.
Jārealizē 3D funkcijas attēlošana.	1. Palaist programmu ar funkcijas algoritmu.	1. Ir redzama 3D funkcjia.	1. Veiksmīgs.
Funkcijai jābūt kustīgai.	1. Palaist programmu ar kustīgu funkciju.	1. Funkcija ir kustīga.	1. Veiksmīgs.

Funkcijas laukumam jābūt savādākā krāsā attiecībā pret funkcijas vērtībām.	1. Palaist programmu ar izstrādātu shaderi.	1. Funkcijas laukums ir savādākā krāsā.	1. Veiksmīgs.
Jāvar aizvērt logu ar (ESC) taustiņu.	1. Aizvērt logu nospiežot (ESC) pogu.	1. Logs aizveras.	1. Veiksmīgs.
Funkcijas laukumam jāmaina savi izmēri attiecībā pret programmas loga lielumu.	1. Mainīt loga platumu. 2. Mainīt loga augstumu.	1. Laukums maina savus izmērus attiecībā pret platumu. 2. Laukums maina savus izmērus attiecībā pret augstumu.	1. Veiksmīgs. 2. Veiksmīgs.
Lai izmainītu kaut ko programmā ir pārāk daudz vietās jāizmaina vieni un tie paši skaitļi.	1. Palaist programmu ar dažādām mainīgā vērtībām.	1. Programma pareizi strādā, nav acīmredzamu kļūdu.	1. Veiksmīgs.
Programmai ir jāstrādā ar ārēji ievadītiem datiem.	1. Palaist programmu ielasot datus no faila.	1. Dati tiek korekti nolasīti no faila.	1. Veiksmīgs.
Uz ekrāna ir jāredz 2D dati 3D vidē.	1. Palaist programmu ar datu vizualizācijas algoritmiem.	1. Dati tiek korekti demonstrēti.	1. Veiksmīgs.
Uz ekrāna atrodas tikai poligonu laukums.	1. Palaist programmu ar pieliktām asīm.	1. Asis pareizi parādās programmā.	1. Veiksmīgs.

Caurspīdīgu poligonu laukumu ir grūti saprast, kopā ar asu līnijām.	1. Palaist programmu ar aizpildītu datu laukumu.	1. Laukums ir labāk parādīts.	1. Veiksmīgs.
Jābūt iespējai apskatīt 3D laukumu no visiem iespējamiem leņķiem.	1. Izmantojot klaviatūras tustiņus apbraukt apkārt un pāri 3D laukumam.	1. Modeli var apskatīt no visām pusēm.	1. Veiksmīgs.

10.Secinājumi

Darba gaidā tika uzlabotas zināšanas par grafisko programmēšanu un par zinātnisko datu vizualizāciju izmantojot dažādas formulas un paņēmienus. Pēc projekta izstrādes autors jūtas daudz pārliecinātāks par savām OpenGL, GLSL un C++ spējām. Spējā izstrādes pieeja bija adekvāta šim projektam, jo tika izpētītas vairākas datu vizualizēšanas metodes kuras izrādījās nederīgas šim projektam.

Darba izstrādes gaitā arī tika gūts priekšstats par vizuālu datu simulāciju veikšanu, izmantojot ārējus datus un laika soļa funkcijas. Tika gūts arī priekšstats par lielu (5000x5000px) bilžu renderēšanu lieliem datiem.

Tika gūts arī priekšstats par nestandarta datu vizualizāciju, piemēram, gravitācijas aku vizualizēšanas metodēm.

11.Izmantotās literatūras saraksts

Interneta resursi:

- https://en.wikibooks.org/wiki/OpenGL_Programming
- https://learnopengl.com/
- http://www.gnuplot.info/
- https://en.wikipedia.org/wiki/Linear_interpolation
- http://www.opengl-tutorial.org/
- https://open.gl/
- https://www.ssec.wisc.edu/~billh/bp/TR.html
- https://www.khronos.org/opengl/

Kvalifikācijas darbs „***OpenGL datu vizualizācijas sistēma***” izstrādāts Latvijas Universitātes Datorikas fakultātē.

Ar savu parakstu apliecinu, ka darbs izstrādāts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: ***Kristaps Veitners***_____ **___.05.2019**.

Rekomendēju darbu aizstāvēšanai

Darba vadītājs: ***Dr.Dat. Jānis Zuters*** _____ **___.05.2019.**

Recenzents: ***Dr.Phys. Modris Bērzonis***

Darbs iesniegts **___.___.2019**.

Kvalifikācijas darbu pārbaudījumu komisijas sekretārs: _____

Darbs aizstāvēts kvalifikācijas darbu pārbaudījuma komisijas sēdē

___.06.2019. prot. Nr. _____

Komisijas sekretārs(-e): _____