|  |  |
| --- | --- |
| RĪGAS TEHNISKĀ UNIVERSITĀTE | |
| DATORZINĀTNES UN INFORMĀCIJAS TEHNOLOĢIJAS FAKULTĀTE | |
| Informācijas tehnoloģijas institūts | |
|  |  |
| Aleksandrs Koroļko | |
| Programmēšanas valodas C alternatīvas datorgrafikā  Bakalaura darbs | |
|  | |
| Vadības informācijas tehnoloģijas katedra  Rīga, Meža iela 1/3 - 400b, LV-1048, Latvija  Tālrunis 67089594 | Zinātniskais vadītājs:  Artūrs Braučs  Programmētājs |
| Rīga – 2019 | |

RĪGAS TEHNISKĀ UNIVERSITĀTE

DATORZINĀTNES UN INFORMĀCIJAS TEHNOLOĢIJAS FAKULTĀTE

Informācijas tehnoloģijas institūts

APSTIPRINU

Vadības informācijas tehnoloģijas katedras vadītājs

Jānis Grabis

201\_. gada \_\_. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Bakalaura darba uzdevums

Students: Aleksandrs Koroļko

1. Bakalaura darba tēma:

latviešu valodā: Programmēšanas valodas C alternatīvas datorgrafikā

angļu valodā: Programming Languages C Alternatives in Computer Graphics

Apstiprināta ar ITI direktora 201\_. gada \_\_. \_\_\_\_\_\_\_\_\_ rīkojumu Nr. \_\_\_\_\_\_\_\_\_\_\_

2. Bakalaura darba nodošanas termiņš: 201\_. gada \_\_. \_\_\_\_\_\_\_\_\_

3. Problēmas nostādne:

Noteikt vai ir C programmēšanas valodas alternatīva datorgrafikā, kurai ir priekšroka OpenGL vidē

4. Bakalaura darba mērķis:

Salīdzināt C programmēšanas valodas alternatīvas OpenGL izmantošanas kontekstā

5. Bakalaura darba uzdevumi:

1. Salīdzināt Rust, GO, V, C programmēšanas valodas paradigmas
2. Iepazīties ar Rust, GO, V sintakse un pamatjēdzieniem
3. Iepazīties ar OpenGL vide
4. Izveidot teksturēta objekta ar gaismas atspīdumu renderēšanas programmu OpenGL vidē katrā valodā
5. Salīdzināt daudzobjektu renderēšanas programmas
6. Izveidot 3D pasaules redaktoru uz visefektīvākā valodā

6. Bakalaura darba konsultants (ja nepieciešams):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(paraksts, datums)

7. Uzdevuma izsniegšanas datums: 201\_. gada \_\_\_. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Bakalaura darba vadītājs: Artūrs Braučs \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(paraksts)

8. Uzdevums pieņemts izpildīšanai: 201\_. gada \_\_\_. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(studenta paraksts)

|  |
| --- |
| RĪGAS TEHNISKĀ UNIVERSITĀTE |
| DATORZINĀTNES UN INFORMĀCIJAS TEHNOLOĢIJAS FAKULTĀTE |
| Informācijas tehnoloģijas institūts |
| Programmēšanas valodas C alternatīvas datorgrafikā |
| Aleksandrs Koroļko |
| Anotācija |

Darba mērķi, uzdevumi un risinājumi. Iegūtie rezultāti un to novērtējums.

Darba apjoms - \_\_\_. lpp., \_\_\_ tabulas, \_\_\_ attēli un \_\_\_ pielikumi.

|  |
| --- |
| RIGA TECHNICAL UNIVERSITY |
| FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY |
| Institute of Information Technology |
| Programming Languages C Alternatives in Computer Graphics |
| Aleksandrs Koroļko |
| Annotation |

Goals of the work, results and their estimations.

The work contains\_\_\_ p., \_\_\_ tables, \_\_\_\_ figures and \_\_\_ appendixes.

|  |
| --- |
| [UNIVERSITĀTES NOSAUKUMS OTRAJĀ SVEŠVALODĀ] |
| [FAKULTĀTES NOSAUKUMS OTRAJĀ SVEŠVALODĀ] |
| [Institūta nosaukums otrajā svešvalodā] |
| [Tēmas nosaukums otrajā svešvalodā] |
| [Studenta vārds, uzvārds otrajā svešvalodā] |
| [„Anotācija” otrajā svešvalodā] |

[Darba mērķi, uzdevumi un risinājumi. Iegūtie rezultāti un to novērtējums. Otrajā svešvalodā.]

[Darba apjoms - \_\_\_. lpp., \_\_\_ tabulas, \_\_\_ attēli un \_\_\_ pielikumi. Otrajā svešvalodā.]

**Saturs**

[IEVADS 7](#_Toc32308813)

[1. Teoretiska Daļa 9](#_Toc32308814)

[1.1. Paradigmas salīdzināšanā 9](#_Toc32308815)

[1.2. Rust, GO Lang, V Lang programmēšanas valodas sintakse un pamatjēdzieni 10](#_Toc32308816)

[1.2.1. Programmēšanas valoda Rust 10](#_Toc32308817)

[SECINĀJUMI 26](#_Toc32308818)

[LITERATŪRA 27](#_Toc32308819)

[PIELIKUMI 29](#_Toc32308820)

IEVADS

Mūsu laikā C valoda skaitās par vienu no populārākajām programmēšanas valodām, pamatojoties uz 2019. gada “Stack Overflow” aptauju[[1]](#footnote-1) un tīmekļa rakstu[[2]](#footnote-2). C valoda skaitās par pirmo vidējo līmeņa valodu, kas savā laikā atviegloja koda rakstīšanu, saprātošanu, paātrināja kopējo programmatūras izstrādāšanu, kā arī izmantojot C valodu bija realizētas UNIX operētājsistēmas un daļēji Windows, iegultās sistēmas API, datubāzes utt., no tā izprast C valodas pamatdarbības ir svarīgi arī mūsdienas, jo vairākas mūsdienas valodas implementē C valodas konstrukcijas, piemērām PHP, Ruby, Python utt., pamatojoties uz rakstiem[[3]](#footnote-3),[[4]](#footnote-4),[[5]](#footnote-5). Bet jā apskatīt lielāko daļu no mūsdienas programmēšanas valodām, tad visi cenšas kļūst par efektīvāko, vieglāko, saprotamāko valodu nekā C valoda, lai aizvietot C valodu un pārņemt priekšroku izmantojot jaunu sintakse, apvienojot vairākas programmēšanas paradigmas, kas palīdz izstrādātājam ātrāk risināt sarežģītās problēmas, savukārt izmantojot C valodu būtu sastopamas vairākas problēmas un var atņemt daudz laika, un samazināties koda kvalitāte, kas ir svarīgie faktori programmatūrās izstrādē, pamatoties uz rakstiem[[6]](#footnote-6),[[7]](#footnote-7). No tā ir iemesls apskatīt vai ir kādas no C valodas alternatīviem, kuriem būs priekšrocībā viena no svarīgākajām datorzinātnes nozarē – datorgrafikā, ar kuru ir iespējams rādīt datora veiktspēju.

Ar datorgrafiku ir saistīta lielāka daļa no datorzinātnes nozare kopš no 1951. gadā [1] un mūsu laikā šajā nozarē bija sasniegtas lielākie rezultāti, piemērām virtuālas un paplašinātas realitātes realizēšana. Viena no svarīgākajām īpašībām datorgrafika – tai jābūt vairākplatformu programmētai un optimizētai, jo lielāka problēma ir saglabāt kvalitāte un veiktspēju vairākās platformās, no tā bija atrasts mūsdienās risinājums datorgrafikā – OpenGL, kura, salīdzinot ar Windows operētājsistēmas datorgrafikas risinājumu DirectX ir labāk vairākas salīdzinājumos [2], un tai piemīt vairākplatformu izmantošana [3]. Arī OpenGL tas ir atklāta grafiska bibliotēka un no tā tai piemīt realizācija vairākās mūsdienīgas programmēšanas valodas, kuri atšķiras tikai sintaksiski, bet realizācijas pamatideja atstājas, ar ko ir ļoti viegli izmantot standartu OpenGL dokumentāciju [4, 12].

Lai noteiktu C valodas alternatīvas, ir jāizvēlas tādas mūsdienīgas programmēšanas valodas, kam ir C programmēšanas valodai līdzvērtīga kompilācijas efektivitātes, pamatdarbības ar datnēm, atmiņas pārvaldība, jo mūsdienas C valoda ir viena no ātrakājam valodām [5], ka arī ir jāizvēlējas tādas alternatīvas, kurās ir realizēta OpenGL bibliotēka.

Pēc vairāku piemēru apskatīšanu pamatojoties uz valodas informāciju “Slant”[[8]](#footnote-8) tīmekļa resursiem, bija izvēlēts Rust, GO un V programmēšanas valodas, kuri skaitās par tuvākajām C valodas mūsdienīgajām alternatīviem, ka arī, tiem ir realizētās OpenGL bibliotēkas.

Lai sasniegt darba mērķi - “Salīdzināt C programmēšanas valodas alternatīvas OpenGL izmantošanas kontekstā”, un noteikt vai ir C programmēšanas valodas alternatīva datorgrafikā, kurai ir priekšroka OpenGL vidē, nepieciešams sekot darba uzdevumiem :

* Salīdzināt Rust, GO, V, C programmēšanas valodas paradigmas – lai noteikt valodas programmēsanas ideju kopumu, kas nosaka rakstīšanas stilu, valodas iespējas un trūkumi, arī noteikt, kādai valodai piemīt vairākas paradigmas;
* Iepazīties ar Rust, GO, V sintakse un pamatjēdzieniem - lai noskaidrot katra valoda, uzticamību, atmiņas manipulēšanas iespējas, valoda pamat idejas un uzdevumus;
* Iepazīties ar OpenGL vide;
* Izveidot teksturētu objektu ar gaismas atspīdumu renderizācijas programmu OpenGL vidē katrā valodā;
* Salīdzināt daudzobjektu renderizācijas programmas – lai noteikt visefektīvāko valodu pamatojoties uz programmas kompilācijas laiku, koda daudzumu, kopējo rakstīšanas paterēto laiku, programmēšanas centieni, programmas uzticamība, ka arī mērīt FPS, atmiņas pateriņu (RAM), izmantojamas GPU un CPU resursi. Priekš precīzākiem rezultātiem, eksperiments jāveic dažādos datoros ar dažādām harakteristikām.
* Izveidot parasto 3D pasaules redaktoru uz visefektīvākā valodā - lai parbautīt programmas kompilācijas laiku, koda daudzumu, kopējo rakstīšanas paterēto laiku, izstrādāšanas centieni, programmas uzticamība, ka arī mērīt FPS, atmiņas patēriņu (RAM), izmantojamas GPU un CPU resursi.

# teorētiskā daļa

Mūsdienu datorgrafika un C ir cieši saistītas ne tikai pamatojoties uz C ātruma, ērta atmiņu pārvaldībā un programmu rakstīšanas elastības dēļ, bet arī tāpēc, ka galvenās ieviešanas un vispāratzīti noteikumi mūsu laikā, piemēram, strādāšana ar objektu virsotnes atmiņas buferiem (VBO), datorgrafikai tika formulēti un ieviesti, izmantojot C, kā arī neskaitāmi paplašinājumi, bibliotēkas, API priekš darbam ar datorgrafiku uz dažādām ierīcēm ar dažādām operētājsistēmām, piemēram OpenGL, tika uzrakstīti arī ar C, kas kopumā dod priekšstats par to, ka C piemīt liela nozīme datorgrafikā.

Pirms pamat programmas izveidošanai katrā izvēlēta programmēšanas valodā, uz kuras pamata būs veidoti galīgie salīdzinājumi, ir svarīgi zināt Rust, GO, V programmēšanas valodas sintakse un pamatjēdzieni, salīdzināt C, Rust, GO un V paradigmas, ka arī iepazināties ar OpenGL pamat būtne, lai izveidot kvalitatīvu programmu katrā valodā, arī ir nepieciešams izplānot programmas salīdzinājumu.

## Paradigmas salīdzināšanā

Ar paradigmām ir iespējams uzzināt kā tiek klasificēta programmēšanas valoda un kādām nolūkam to ir jāizmanto. Salīdzināšanai tiek izveidota tabula (1.1 tabula), kurā ir paradītās izvēlētas darba programmēšanas valodas ar visām paradigmām (“+” - “piemīt”, “+/-” – daļēji piemīt vai ir sava interpretācija, “-” – “nepiemīt”, “?” – nav informācija), kuras bija minētās programmēšanas valodas Rust, GO Lang, V Lang dokumentācijā [9-11] un tīmekļa rakstos[[9]](#footnote-9),[[10]](#footnote-10),[[11]](#footnote-11) par kopēju informāciju par paradigmāmH.

1.1 tabula

**Izvēlētas programmēšanas valodas paradigmas**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | C | GO | V | Rust |
| Procedurāla | + | + | + | + |
| Imperatīva | + | + | + | + |
| Objektorientēta | - | +/- | +/- | +/- |
| Vienlaicīga  (Concurrent) | - | + | + | +/- |
| Strukturētā  (Structured) | + | + | + | + |
| Ģenerisks  (Generic) | + | - | + | + |
| Funkcionāla | +/- | +/- | +/- | + |
| Refleksīva  (Reflection) | - | - | ? | +/- |

Ir redzams, kā lielāka daļa no paradigmas piemīt Rust, kas dod priekšrocību virs GO un C, jo ir iespējams izmantojot vienu valodu risināt vairākas problēmas ar vairākiem paradigmas idejām (multi-paradigmas valodas), bet tas varētu kļūst par problēmu, kad tiek izmantotas vairākas paradigmas idejas vienā risinājumā, jo tiek pazemināta koda kvalitāte un tās funkcionalitāte, savukārt analizējot V nebija atrasta informācija par refleksīvu paradigmu, no kas nav iespējams pilnvērtīgi salīdzināt to ar citām valodām.

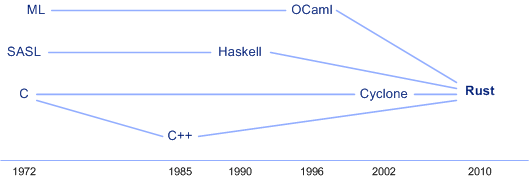
Protams, ka C valodai piemīt mazāka daļa no paradigmas, jo tā ir viena no pirmajiem piemēriem imperatīva paradigmā, uz kura pamatā tika izveidotas vairākas mūsdienīgi zināmās paradigmas pamatojoties uz 1. pielikuma attēlu.

## Rust, GO, V programmēšanas valodas sintakse un pamatjēdzieni

Katra programmēšanas valoda tiek veidota, lai palīdzēt ātrāk un drošāk atrisināt vairākas specifiskas problēmās datorzinātnes un informācijas tehnoloģijas nozarē, kur citās valodās būs sastopamas ar vairākām problēmām, no tā ir svarīgi zināt katrā izvēlēta programmēšanas valoda priekšrocībās, pamatojaties uz tās sintakse un pamatjēdzieniem, kā arī nodefinēt valodas uzticamību.

### Programmēšanas valoda Rust

Rust programmēšanas valodu uzskatīta par C tipa programmēšanas valodu, kuru projektējis Graydon Hoare ar Dave Herman un Brendan Eich un izstrādāja “Mozilla”, kurā pamatā tiek lietots multi-paradigmas piegājiens, kas savieno sevī vairākas programmēšanas idejas, kā tiek minēts IBM rakstā[[12]](#footnote-12). Rust tuvākie priekšgājēji ir OCaml, Haskell, C++ un Cyclone programmēšanas valodas, no kuriem bija pārņemtas vairākas sintakses un datnes manipulēšanas konstrukcijas un idejas (1.1 att.[[13]](#footnote-13)), kā piemērs tuva JavaScript funkcijas “async await” konstrukcija, paņemts no OCaml, polimorfisma deklarēšana, “trait” datu klases, saistītie datnes tipi, paņemts no Haskell, utt.



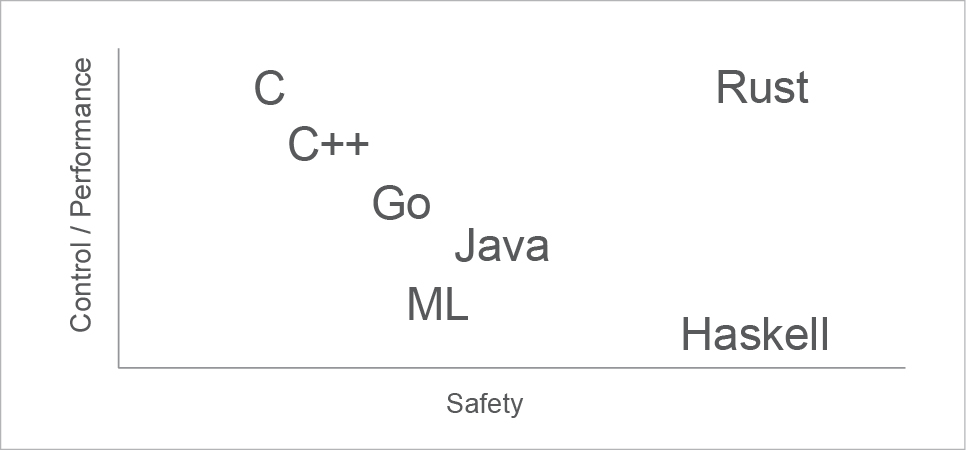
1.1 att. **Rust priekšgājēji**

Rust piemīt savs kompilators “rustc”[[14]](#footnote-14), kuram ir nepieciešams C kompilators, piemēram “MinGW”, un vienkāršo projekta sastādīšana, jo ir nepieciešams norādīt galveno projekta “crate root” failu un “rustc” pats pārbauda visas saistības ar projektu moduļiem un bibliotēkām. Tas atšķiras no tā, kā ir jāizmanto C kompilatoru, kur katram projekta failam ir nepieciešams izsaucat kompilatoru un pēc tam saistāt visu kopā, kas palielina kopējo izstrādāšanas laiku.

Arī Rust piemīt savs paketes menedžeris “Cargo”[[15]](#footnote-15), ar kuru ir iespējams ātri un ērti pievienot dažādas koda bibliotēkas/paketes projektā, kontrolēt projekta versiju un paketes atkarībās, sastādīt projektu ar izpildāmo failu priekš dažādām operētājsistēmām, utt., kas ļauj koncentrēties tikai uz projekta izpildīšanu un vieglāk izmantot nepieciešamas bibliotēkas/paketes savā projektā, kas paātrina kopējo izstrādāšanas laiku un dod uzticamību par kopējo projektu.

#### Pamatjēdzieni

Pamatojoties uz Rust dokumentāciju [9] un uz tīmekļa rakstiem[[16]](#footnote-16),[[17]](#footnote-17), Rust programmēšanas valoda pamatjēdziens apvienot zema līmeņa un augšēja līmeņa programmēšanas valodas jēdzienus, kas rezultātā iekļauj sevī izpildes ātrumu un drošību kā ir paradīts attēlā zemāk (1.2 att.[[18]](#footnote-18)), bet no tā var arī zaudēties programmas kopējais kompilācijas laiks, kā tiek minēts oficiālā vietnē[[19]](#footnote-19).



1.2 att. **Rust priekšrocības grafiskais salidzinājums**

Otrais Rust pamatjēdziens ir atmiņas efektīva un droša pārvaldība, kura iekļauj sevī atmiņas steka glabāšana, tieša piekļuve atmiņai un manuāla atmiņas pārvaldība (“unsafe” režīmā metodes), un izmantošana, kas ir svarīgi datorgrafikā, jo ir nepieciešams pēc iespējas ātrāk piekļūst, daļēji vai pilnīgi mainīt, dzēst nepieciešamās datnēs buferus un nezaudēt laiku uz datnes eksistēšanas pārbaudīšanu, ar ko var palīdzēt Rust.

Kā piemērs, Rust vidē piemīt specifiska datnes kontrolēšana - mutabilitāte, kas palīdz darboties ar atmiņu drošāk un tīrāk, un viens no tās piemēriem ir sadalīt mainīgus uz nemainīgiem, vienā programmas kontekstā izmantojama “konstante”, kuras vērtību nav iespējams pārdefinēt un tās vieta atmiņā tiek atbrīvota pēc programmas izmantošanas konteksta nobeiguma, kas rezultātā neaizņem vietu kaudzē, un mainīgiem, kurus jādefinē ar “mut” pamat vārdu un ir iespējams pārdefinēt tās vērtību, arī no tā, ka Rust vidē ir realizēts pie katra resursa tikai viena tieša saite, kas nodrošina no datnes vērtības zaudēšanas, nav iespējams izmantot vienu un tādu pašu mainīga vērtību dažādās vietās vienlaicīgi, tabulā lejā (1.2 tabula.) ir paradītas vektora mainīga “v” vērtība saistījumi ar cita mainīgā “v2” un “v” vektora vērtība pārnēsāšana citā kontekstā, funkcijā “take”, kas rezultātā atgriež pirms programmas palaišanas kļūdu “use of moved value: `v`”, jo vērtība ir pārvietotā citā mainīgā un citā programmas īpašumtiesības kontekstā.

1.2 tabula

**Datnes saistījuma kļūdainie piemēri**

|  |  |
| --- | --- |
| Piemērs ar mainīgiem | Piemērs ar funkciju |
| let v = vec![1, 2, 3];  let v2 = v;  println!("v[0] is: {}", v[0]); | fn take(v: Vec<i32>) {  // funkcijas loģika  }  let v = vec![1, 2, 3];  take(v);  println!("v[0] is: {}", v[0]); |

No tā, lai nezaudēt mainīgas vērtību vairākas īpašumtiesības kontekstā ir pieņemts strādāt izmantojot adreses un rādītājus, kur rādītāji apkopo standarta C valodas rādītājus bez papildu marķēšanas vai izpildlaika pārbaudēm, no kas Rust vidē nav vajadzīgs drāzu savācējs, kas paātrina kopdarbību, jo visa analīze (kļūdas un brīdinājumi par neizmantotam mainīgām, paketēm, utt.) tiek veikta ar “rustc” kompilatoru programmas sastādīšanas laikā, tādējādi izpildes laikā radot “nulles izmaksas” un bez atmiņas noplūdēm, kā tiek minēts tīmekļa rakstā[[20]](#footnote-20) un dokumentācijā [9] (79. lpp.), kas ir svarīgi datorgrafikā, jo Rust izmanto datora resursus droši un optimāli, kas ļauj strādāt uzticamīgāk ar datorgrafikas daudzobjektu renderizācijas programmām datnēm.

#### Sintakse

Rust sintakse apvieno sevī lielāko daļu no C un OCaml sintakses, kas rezultāta, ļauj ātrāk iedziļināties C valodas datnes un atmiņas pārvaldība programmētājiem, kuri neizmantoja C tipa valodas, bet ir zināšanās, piemērām, JavaScript valodā. Rust sintaksē ir kombinētās vairākas atslēgvārdi, kas rezultātā veido lasāmāko kodu un ļauj vieglāk kontrolēt vāja tipa mainīgus, gan arī “unsafe” režīmā, struktūras datu kolekcijas, testēšanas pamatfunkcijas, arī ir uzlabotas un vienkāršotas vairākas C programmēšanas valodas sintakses konstrukcijas, kas dod kopējo uzticamību datnes pārvaldībai Rust valodai, kas ir labs iemesls izmantot to datorgrafikā.

Tabulā (1.3 tabula) ir paradītās bāziskas sintakses piemēri ar tās semantiku.

1.3 tabula

**Sintakses piemēri ar semantiku**

|  |  |
| --- | --- |
| **Sintakse** | **Semantika** |
| let mut x = 5;  x = 10; | Izveidot mainīgo x un piešķirt 5;  Piešķirt 10 pie mainīga x; |
| let x: i32 = 8;  {  println!("{}", x); //Izvada "8"  let x = 12;  println!("{}", x); //Izvada "12"  }  println!("{}", x); // Izvada "8" | Izveidot mainīgo x ar tipu int32 un piešķirt 8;  Atvērt tvērumu;  Izvadīt x; Komentārs “Izvada "8"”  Piešķirt 12 pie x;  Izvadīt x; Komentārs “Izvada "12"”  Ciet tvērumu;  Izvadīt x; Komentārs “Izvada "8"” |
| fn add\_one(x: i32) -> i32 {  x + 1  } | Funkcija “add\_one” ar parametru x int32 tipa un int32 atgriešanās tipu;  Funkcija atgriež x + 1; |
| let y = if x == 5 { 10 } else { 15 }; | Piešķir 10 pie y, jā x ir vienāds ar 5, citādi piešķir 15; |
| struct Point {  x: f64,  y: f64  }  fn main ()  {  let mypoint =  Point {x: 1.0, y: 2.0};  println!(mypoint.x);  } | Deklarēt struktūru ar nosaukumu “Point”, kurā ir lauks x un y ar datu tipu float64;  Funkcijā “main” izveidot jaunu objektu “mypoint” ar “Point” tipu, piešķirt x laukam 1.0, y laukam 2.0;  Izvadīt objekta “mypoint” x lauka vērtību. |
| fn addsub(x: i32, y: i32)  -> (i32, i32)  {  (x + y, x - y)  }  fn main () {  let mut (add, sub)  = addsub(4, 2);  } | Izveidot funkciju “addsub”, ar parametriem x un y, ar int32 tipu, kas atgriež divas int32 vērtībās, kur pirmā vērtība ir x + y un otra vērtība ir x – y;  Funkcijā “main” piešķirt funkcijas “addsub” ar parametriem 4 un 2, pirmo vērtību mainīgajā “add” un otro vērtību mainīgajā “sub”; |

Arī datorgrafikā ir nepieciešamas datu tipa atmiņas piešķišanas kontrolēšana, lai samazināt atmiņas zaudējumus priekš datnes vienībām dažādas sistēmas arhitektūras, ar kuras pārveidošanu Rust vidē nodarbojas “rustc” kompilators, un programmētājām Rust valodā piedāvā izmantot dažādas primitīvas datu tipus (1.4 tabula.), kurus kopā ar struktūras tipiem ir iespējams viegli deklarēt statiski, dinamiski, precīzi (explicit), netieši (implicit) utt., pārveidot viens otrā, kas rezultāta dod optimālo sistēmas resursus izmantošanu.

1.4 tabula

**Rust primitīvie datu tipi**

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| bool | Būla tips |
| char | Simbola tips |
| i8 | 8 bitu parakstīts vesels skaitlis |
| i16 | 16 bitu parakstīts vesels skaitlis |
| i32 | 32 bitu parakstīts vesels skaitlis |
| i64 | 64 bitu parakstīts vesels skaitlis |
| isize | Rādītāja lieluma parakstīts vesels skaitlis |
| u8 | 8 bitu neparakstīts vesels skaitlis |
| u16 | 16 bitu neparakstīts vesels skaitlis |
| u32 | 32 bitu neparakstīts vesels skaitlis |
| u64 | 64 bitu neparakstīts vesels skaitlis |
| usize | Rādītāja lieluma neparakstīts vesels skaitlis |
| f32 | 32 bitu peldošā komata tips |
| f64 | 64 bitu peldošā komata tips |
| array | Fiksēta izmēra masīvs, apzīmēts ar [T; N], elementu tipam T un nenegatīvajam konstantam izmēram N. |
| slice | Dinamiska datnes kolekcija |
| str | Dinamiska simbolu virkne (simbolu kolekcija) |
| tuple | Ierobežota neviendabīga secība  Piemēram ("hello", 42, "world", [3,6,9]) |

### Programmēšanas valoda GO

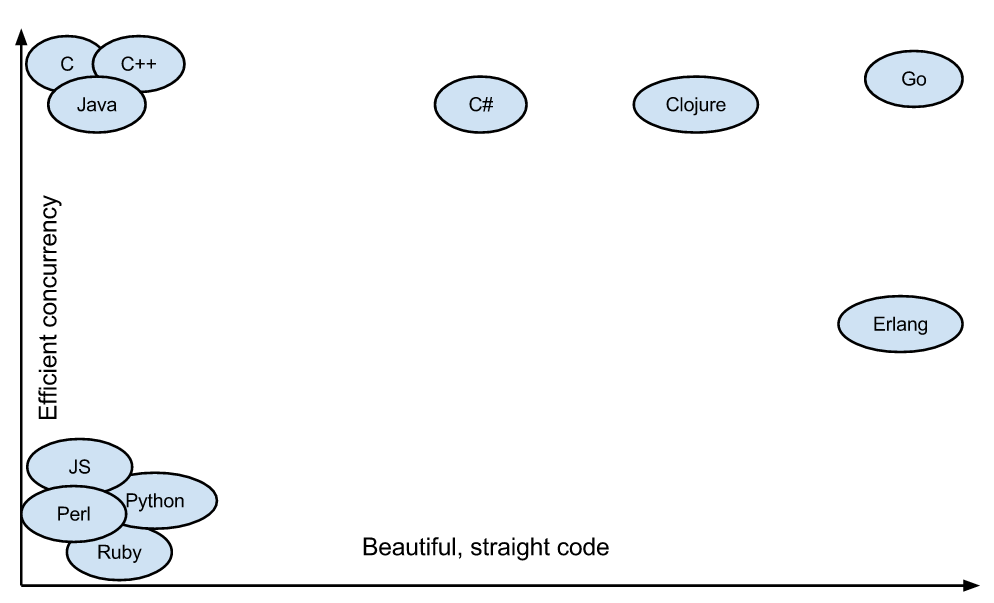
GO ir C (C++), Java un Python programmēšanas valodas apvienojums, kurš tiek izstrādāts ar “Google” 2007 gadā, kā tiek minēts GO programmēšanas valoda atskaite[[21]](#footnote-21), kurā pamatideja saistīta ar optimālo resursu izmantošanu vairākseržu procesorus un neizmantojot virtuālo mašīnu slāņu, piemēram JVM (Java Virtual Machine), pārveidot cilvēkam lasāmu kodu uzreiz uz bināru kodu procesora vidē, kā tiek minēts tīmekļa raksta[[22]](#footnote-22), kas paātrināta kopdarbību un uzlabo datnes pārvaldību starp vairākiem datora procesiem, kas varētu labi ietekmēt uz datorgrafikas programmas kopdarbību, tieši daudzobjektu renderizācijas programmām, kad ir nepieciešams parvaldīt vairākas datorgrafikas objektu datnes vienlaicīgi/paralēli.

GO piemīt savs kopmilators “gc”[[23]](#footnote-23), kurš uzrakstīts uz GO, bet no tā, ka pirmas “gc” versijas bija uzrakstītas pamatojoties uz C un nevisas GO bibliotēkas strāda pamatojoties tikai uz GO, piemēram standartas OpenGL bibliotēkas “GL” un “GLFW” GO implementācijā, ir nepieciešams C kompilators sistēmā, piemēram “MinGW”, ar visām nepieciešamām “.h” un “.dll” bibliotēkas failem. GO “gc” kompilatoram tāpat, kā “rustc”, piemīt vienkārša projekta sastādīšana, jo ir nepieciešams norādīt galveno projekta “main.go” failu un kompilators pats pārbauda visas saistības ar projektu importētiem moduļiem un bibliotēkām, un brīdina par neizmantotiem koda elementiem, kas ļauj turēt projektus tīrāk un nezaudēt datora resursus.

#### Pamatjēdziens

GO valoda pamatjēdziens ir novērst lēnumu un neveiklību daudzkodolu procesora sistēmās, tādējādi padarot procesus produktīvāk un mērogojamāk, un atvieglot, nodrošināt, paātrināt pavedienu izveidošanu, samazināt tās glabāšanās vietās daudzumu kaudzē, kas tiek realizēts ar izsekojamām datora resursu vadībām - iebūvētiem pavadoņiem “goroutines”, kā tiek mīnēts un aprakstīts oficiāla apmācības veidnē[[24]](#footnote-24),[[25]](#footnote-25). Arī ir realizēts savs drazu savācējs, kas ļauj neatstāt kaudzē neizmantotas rezervētās datnes, bet ietekmē uz datora resursiem programmas izpildes laikā. Kā piemērs, “Google” izmanto GO servera puses pieprasījumu apstrādāšanai un vairākos projektos, gan arī vecos, kur izmantotas lielas programmatūras sistēmas ar pavedieniem, kas arī ir pielietojams datorgrafikas kontekstā, tieši OpenGL, kad ir nepieciešams izmantot pavedienus priekš katra renderēta objekta un nezaudēt lauku uz secīgu datnes apstrādāšanu.

Kā tiek parādīts attēlā (1.3 att.[[26]](#footnote-26)), GO skaitās par efektīvāku vienlaicīgumā un vienu no caurspīdīgākām koda sintakse programmēšanas valodām, bet no tā, ka GO sintakse atšķiras no C programmēšanas valodas sintakse ir vajadzīga koda pārtulkošana uz C kompilatora vidē, kas ietekmē uz programmas kompilācijas laiku.



1.3 att. **Vienlaicīgumā un koda caurspīdīgumā valodas grafiks**

#### Sintakse

GO sintakse ir vairāk pietuvināta pie Pascal nekā C, bet tāpat ir izmantotas vairākas C konstrukcijas. Lielāka atšķirība no citām C valodām ir tāda, ka GO sintakses vidē ir tikai 25 atslēgvārdu, bet piemērām C99 ir 37 un C++11 ir 84 atslēgvārdu skaits un šīs skaits palielinās, un daļēji GO semantika ir atvieglota salīdzinot ar C valodas tipa semantiku, piemērām: nav rādītāja aritmētikas, nav netieša skaitliska konvertācija, masīva robežas vienmēr tiek pārbaudītas utt., kas ļauj saprasts GO sintakse ātrāk un no tā, ļauj jauniem programmētājiem ātrāk iedziļināties C veida programmēšanas valodas vidē. Bet no tās atvieglošanas GO nav realizēta tik efektīva atmiņas pārvaldība, kā Rust, no tā izmantojot GO datorgrafikā būs redzamas atmiņas lielāks patēriņš priekš katra datorgrafikas objekta datnes bufera, kas var ietekmēt uz kopējo datora resursu patēriņu, izraisīt atmiņas noplūde, gan arī ietekmēt uz CPU izmantojamiem resursiem, jo nepieciešams apstrādāt, parbaudīt datnes un pārveidot daļu no GO atvieglota semantika C kompilatora vidē.

Tabulā (1.5 tabula) ir paradītās bāziskas sintakses piemēri ar tās semantiku.

1.5 tabula

**Sintakses piemēri ar semantiku**

|  |  |
| --- | --- |
| **Sintakse** | **Semantika** |
| var x string = "Hello World"  fmt.Println(x) | Izveidot mainīgo x String tipa un piešķirt “Hello World”;  Izvadīt x mainīga saturu konsolē; |
| var arr [5]int  x[4] = 100  fmt.Println(x) | Izveidot piecu int tipa elementu masīvu arr;  Pievienot pie arr piektā elementa 100;  Izvadīt masīvu arr konsolē; |
| func add\_one(x int) int {  return x + 1  } | Funkcija “add\_one” ar parametru x int tipa un int atgriešanās tipu;  Funkcija atgriež x + 1; |
| func main() {  addsub := func(x, y int) int {  return x + y, x - y  }  fmt.Println(addsub(5,6))  } | Izveidot funkciju ar parametriem x un y, int tipa, kura atgriež divas int vērtībās, kur pirmā vērtība ir x + y un otra vērtība ir x – y, main funkcijā;  Ievadīt jaunajā funkcijā 5 un 6 un izvadīt konsolē; |
| type Point struct {  x, y float64  }  func main() {  var mypoint Point  mypoint := Point{1.0, 2.0}  fmt.Println(mypoint.x)  } | Deklarēt struktūru ar nosaukumu “Point”, kurā ir lauks x un y ar datu tipu float64;  Funkcijā “main” izveidot jaunu objektu “mypoint” ar “Point” tipu, piešķirt x laukam 1.0, y laukam 2.0;  Izvadīt objekta “mypoint” x lauka vērtību. |
| func example(n int) {  for i := 0; i < n; i++ {  fmt.Println(i)  }  }  func main() {  go example(5)  } | Izveidot metode example ar paramentru n int tipa, kur ar ciklu izvada visus ciparus no 0 līdz n;  Funkcijā main izmantojot pavedienu izsaukt metode example ar n vērtību 5; |

<https://kuree.gitbooks.io/the-go-programming-language-report/content/2/text.html>

<https://golang.org/doc/faq#Is_Go_an_object-oriented_language>

GO vidē ir implementētas vairākas primitīvas datu tipi (1.6 tab.), kopā ar struktūras tipiem, kurus ir iespējams viegli deklarēt statiski, precīzi (explicit), utt., bet nav iespējams deklarēt dinamiski un netieši (implicit), no kās rodas nepieciešamība tērēt atmiņu uz vairākiem tieši tipizētiem mainīgiem, jā ir vajadzīgi implementēt viena kodu loģiku priekš dažādiem datu tipiem, kas var iekļaut uz kopējo programmas atmiņas patēriņu, bet tāpat GO valodu ir uzticamība lietot datorgrafikā, jo neskatoties uz ne tik labi optimizēto atmiņas un resursu pārvalde, nekā Rust, GO valodā ir iespējams ātri un viegli realizēt datorgrafikas daudzobjektu datnes pārvaldību vienā laikā, izmantojot “goroutines”.

1.6 tabula

**GO primitīvie datu tipi**

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| bool | Būla tips |
| char | Simbola tips |
| **int8 vai byte** | 8 bitu parakstīts vesels skaitlis |
| **int16** | 16 bitu parakstīts vesels skaitlis |
| **int32 vai rune** | 32 bitu parakstīts vesels skaitlis |
| **int64** | 64 bitu parakstīts vesels skaitlis |
| **int** | 32 vai 64 bitu parakstīts vesels skaitlis |
| uint8 | 8 bitu neparakstīts vesels skaitlis |
| uint16 | 16 bitu neparakstīts vesels skaitlis |
| uint32 | 32 bitu neparakstīts vesels skaitlis |
| uint64 | 64 bitu neparakstīts vesels skaitlis |
| uint | 32 vai 64 bitu pa neparakstīts vesels skaitlis |
| **uintptr** | Tas ir neparakstīts vesels skaitlis. Tās izmērs nav noteikts, bet var turēt visus rādītāja vērtības bitus. |
| float32 | 32 bitu peldošā komata tips |
| float64 | 64 bitu peldošā komata tips |
| **complex64** | Sarežģīti skaitļi, kas satur float32 kā reālu un iedomātu sastāvdaļu. |
| **complex128** | Sarežģīti skaitļi, kas satur float64 kā reālu un iedomātu sastāvdaļu. |
| array | Fiksēta izmēra masīvs, apzīmēts ar [T; N], elementu tipam T un nenegatīvajam konstantam izmēram N. |
| slice | Dinamiska datnes kolekcija |
| string | Dinamiska simbolu virkne (simbolu kolekcija) |
| map | Masīva tips, kurā datnes vērtības atslēgs varētu būt jebkura tipa |

### Programmēšanas valoda V

V – tas ir uz GO uzrakstīta un iedvesmotā ar Rust, Swift, Oberon, jauna programmēšanas valoda, kura apvieno sevī vairākas C tipa programmēšanas valodas pamatjēdzienus un priekšrocības, un pievieno jaunas C tipa valodas iespējās, kā tiek paradīts tabulā (1.7 tabula) lejā.

1.7 tabula

**Valodas iespējas un priekšrocības**

|  |  |
| --- | --- |
| **Priekšrocības un iespējas** | **Valodas** |
| Ātra kompilācija | D, Go, Delphi |
| Vienkāršība un uzturamība | Go |
| Lieliska veiktspēja, salīdzinot ar C un nulles izmaksas | D, Delphi, Rust |
| Drošība | Rust |
| Viegls vienlaicīgums | Go |
| Šķērskompilācija | Go |
| Laika koda ģenerēšana un kompilēšana | D |
| Mazais kompilators ar nulles atkarībām | - |
| Nav globāla stāvokli | - |
| “Hot code” pārlādēšana | - |

Pēdējās trīs rindas nav minētas valodas, ar to tiek paradīts, ka pagaidām nav tādas C valodas ar sekojošam iespējām un priekšrocībām, bet V iekļauj sevī arī tos.

Pagaidām V ir Beta versijas stāvoklī un pilna v1.0 versija tiek ieplānotā 2020. gadā beigās, bet jau V tiek izmantots un testēts vairākas servera daļa, WEB, datorgrafikas, kross platformas, GUI programmatūrās, ka arī V vidē tiek piedāvāts C pārtulkotājs, kurš var pārtulkot kodu no V uz C un otrādāk.

#### Pamatjēdziens

Galvenais pamatjēdziens V ir kompilatora paātrināšana un mazāka vietu aizņemšana, kā tiek paradīts tabulā (1.8 tabula) lejā, un ir redzams, ka V kompilators ar pamat bibliotēkām aizņem vismazāk atmiņu un tiek uzcelts sistēmā ātrāk nekā citi, bet V kompilators ir bāzēts uz C kompilatora un izmanto vairākas C bibliotēkas, un no tā ir nepieciešams instalēt sistēmā arī C kompilatoru, piemēram “MinGW”.

1.8 tabula

**Kompilatora un pamat bibliotēkas aizņemta vieta un uzcelšanas laiks salīdzināšana**

|  |  |  |
| --- | --- | --- |
| Kompilators | Aizņemta vieta | Uzcelšanas laiks |
| GO Lang | 525 MB | 1m 33s |
| Rust | 30 GB | 45m |
| GCC | 8 GB | 50m |
| C Lang | 90 GB[[27]](#footnote-27) | 25m |
| Swift | 70 GB[[28]](#footnote-28) | 90m |
| V Lang | < 2 MB | 0.09s – 1s |

V kompilators tāpat, kā GO un Rust kompilatori analīze programmas kodu sastādīšanas laikā un brīdina par kļūdām, iespējamām izpildīšanas laikā kļūdainam vietām un brīdina par neizmantotam mainīgām, paketēm, utt., lai turēt projektu tīrāk un nezaudēt datora resursus un neizmantojamām elementam.

Otrs V Lang pamatjēdziens ir uzlabot un apvienot vienā vidē vairākas C tipa valodas iespējas un pamatjēdzienus, un kļūst ātrāk, vieglāk, drošāk nekā C, un jau tagad V ļauj kompilēt ātrāk nekā citas C tipa valodas, jo kā tiek minēts oficiālā tīmekļa lapā [11], V var kompilēt no 100 tūkstošiem līdz 1.2 miljoniem koda rindas sekundē (uz datora ar sekojošo dzelzi : Intel i5-7500 @ 3.40GHz, SM0256L SSD), jo programma tiek kompilēta bez jebkādām atkarībām uzreiz bināra kodā.

V ir iemesls mēģināt izmantot datorgrafikas programmas izstrādē, ne tikai no tās ātruma un jau izveidotas sākum bibliotēkas, bet arī, jo tiek izmantots Rust atmiņas pārvaldes elastīgums un drošums, gan arī izmantots GO viegla un ātra un efektīva pavedienu izveidošana un izmantošana, kas rezultātā var dod drošu un ātru datorgrafikas programmu, kurā tiek optimāli ņemtas nepieciešami resursi katram pavedienam, kurā tiek apstrādāts atsevišķs datorgrafikas objekts ar viesiem nepieciešamiem datnes buferiem. Bet no tā, ka V ir Beta versijā, var radīties gadījumi ar atmiņas noplūde, jo galīgs variants ar atmiņas pārvalde pagaidām nav gatavs, kā tiek minēts oficiālā dokumentācijās tīmekļa lapā [13], no tā V pagaidām neizmanto produkcijas izveidošanai.

V ir pirmais C valodas piemērs kurā ir iekļauta “Hot code” iespēja, ar kuru nav nepieciešams pārlādēt visu projektu, kad bija izveidotas mazas koda izmaiņas, kuras neietekmē uz kopējo projekta struktūru, un tādas izmaņas un tiek kompilētas uzreiz, kas ir pielietojams datorgrafikas programmas, lai testēt objekta elementus, piemērām tekstūras, objekta ēnotāju komandas utt.

#### Sintakse

V sintakse ir apvienojums no GO, Rust, Swift, Oberon un C++ sintakses, kur vairākas sintakses konstrukcijas ir uzlabotas, paātrinātas un atvieglotas. Kopumā V sintakse apvieno sevi GO vienkāršību, ātras un drošas Rust atmiņas pārvaldības iespējas un konstrukcijas, un citus C tipa valodas darbspēja uzlabošanas risinājumus.

Tabulā (1.9 tabula) ir paradītās bāziskas sintakses piemēri ar tās semantiku.

1.9 tabula

**Sintakses piemēri ar semantiku**

|  |  |
| --- | --- |
| **Sintakse** | **Semantika** |
| mut age := 5  age = 10 | Izveidot mainīgo x un piešķirt 5;  Piešķirt 10 pie mainīga x; |
| mut arr := ['Ieva']  arr << 'Peteris'  println(arr.len)  println('Henrijs' in arr) | Izveidot masīvu “arr” ar “Ieva” vērtību;  Pievienot masīvā “Peteris”;  Izvadīt konsolē masīva garumu;  Izvadīt konsolē pārbaudījumu vai ir “arr” masīvā elements “Henrijs”; |
| y := if x == 5 { 10 } else { 15 } | Piešķir 10 pie y, jā x ir vienāds ar 5, citādi piešķir 15; |
| struct Point {  x f64  y f64  }  fn main () {  p := Point{  x: 1.0  y: 2.0  }  println(p.x)  } | Deklarēt struktūru ar nosaukumu “Point”, kurā ir lauks x un y ar datu tipu float64;  Funkcijā “main” izveidot jaunu objektu “mypoint” ar “Point” tipu, piešķirt x laukam 1.0, y laukam 2.0;  Izvadīt objekta “mypoint” x lauka vērtību. |
| fn addsub(x, y int) (int, int) {  return x + y, x - y  }  fn main () {  add, sub := addsub(4, 2)  } | Izveidot funkciju “addsub”, ar parametriem x un y, ar int32 tipu, kas atgriež divas int32 vērtībās, kur pirmā vērtība ir x + y un otra vērtība ir x – y;  Funkcijā “main” piešķirt funkcijas “addsub” ar parametriem 4 un 2, pirmo vērtību mainīgajā “add” un otro vērtību mainīgajā “sub”; |
| fn sqr(n int) int {  return n \* n  }  fn numbers(n int) int {  mut sum := 0  mut i := 0  for i <= n {  sum += i  i++  }  return sum  }  fn run(value int, op fn(int) int) int {  return op(value)  }  fn main() {  println(run(5, sqr))  println(run(5, numbers))  } | Izveidot funkciju “sqr”, ar parametru “n” int tipa, kas atgriež int tipa rezultātu n \* n;  Izveidot funkciju “numbers\_sum”, ar parametru “n” int tipa, kas atgriež int tipa summu no 0 līdz n;  Izveidot funkciju “run”, ar parametriem “value” (int tipa), “op” funkcijas nosaukums ar vienu int parametru, kas atgriež int vērtību, kurā tiek izsauktas funkcijas un atgrieztas tās vērtības;  Funkcijā “main” izvadīt rezultātus “sqr” un “numbers\_sum”, izmantojot funkciju “run”, kur “value” ir vienāda ar 5; |

Tā, kā V atmiņas pārvalde ir cieši saista ar Rust, tad visas datu tipa atmiņas piešķišanas kontrolēšana V kompilatora ir pārņemta no “rustc” Rust kompilatora un no tā, V valodā ir iespējams izmantot dažādas primitīvas datu tipus (1.10 tabula) un kopā ar struktūrām viegli deklarēt statiski, dinamiski, precīzi (explicit), netieši (implicit) utt., pārveidot viens otrā, kas rezultāta dod optimālo sistēmas resursus izmantošanu, bet no tā, ka V ir Beta versijā, nav tāda paša uzticamība, kā piemērām GO vai Rust, optimālai datnes pārvaldībai.

1.10 tabula

**V primitīvie datu tipi**

|  |  |
| --- | --- |
| Nosaukums | Apraksts |
| bool | Būla tips |
| **i8** | 8 bitu parakstīts vesels skaitlis |
| **i16** | 16 bitu parakstīts vesels skaitlis |
| **int** | 32 bitu parakstīts vesels skaitlis |
| **i64** | 64 bitu parakstīts vesels skaitlis |
| byte | 8 bitu neparakstīts vesels skaitlis |
| u16 | 16 bitu neparakstīts vesels skaitlis |
| u32 | 32 bitu neparakstīts vesels skaitlis |
| u64 | 64 bitu neparakstīts vesels skaitlis |
| rune | apzīmē Unicode koda punktu |
| **uintptr** | Tas ir neparakstīts vesels skaitlis. Tās izmērs nav noteikts, bet var turēt visus rādītāja vērtības bitus. |
| f32 | 32 bitu peldošā komata tips |
| f64 | 64 bitu peldošā komata tips |
| byteptr | Vesels bita radītāja tips. Ir pielietojams byte\* un char\* |
| voidptr | Dinamisks radītāja tips |
| array | Dinamiska izmēra masīvs, apzīmēts ar [N]T, elementu tipam T un nenegatīvajam konstantam izmēram N (var arī būt tukšs). |
| string | Dinamiska simbolu virkne (simbolu kolekcija) |
| map | Masīva tips, kurā datnes vērtības atslēgs varētu būt jebkura tipa, (pagaidām ir realizēts ar string un vesela datu tipiem) |

## OpenGL pamati

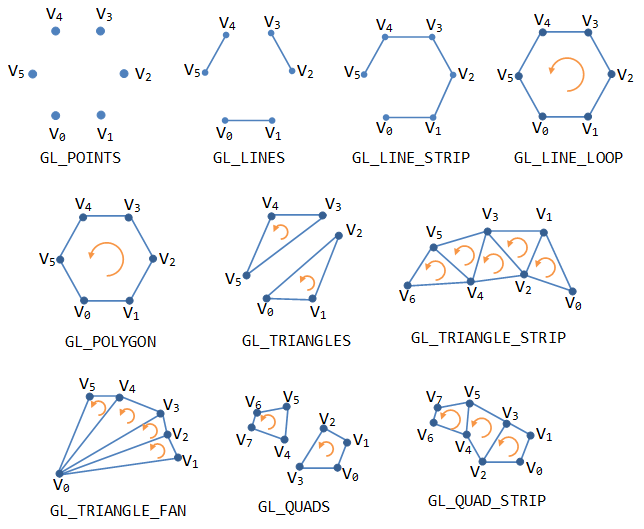
OpenGL (“Open Graphics Library”) ir API (lietojumprogrammu interfeiss) datorgrafika, kurš sastāv no vairāku simtu funkciju un procedūru kopuma, kas ļauj norādīt ēnotāju programmas, objektus un darbības, kas saistītas ar augstas kvalitātes grafisko attēlu veidošanu. Sākot no OpenGL 4.3 versijas gandrīz visas operācijas, matemātiskas darbībās, datu un attēla kadra buferi glabāšana tiek realizēta ar GPU (grafiskais procesors) atmiņu un ēnotāju programmas izpilde tiek pārnesta uz GPU procesiem, kas kopumā paātrina grafiskas zīmēšanas procesus, bet arī tiek atstāta CPU izmantošana, kad GPU veiktspēja ir maza.

Šajā nodaļā ir aprakstīts OpenGL izpildes modelis, globāla telpa attēla transformācijas, ēnotāju programmas jēdziens un objekta apgaismojums.

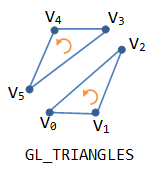
### Izpildes modelis

OpenGL izpildes modelis nodrošina tiešu 3D un 2D grafikas pamatfunkciju kontroli, kā piemērām, primitīvas manipulēšana, parametru precizēšanu ēnotāju programmās, kropļojumnovērse utt., bet OpenGL nenodrošina līdzekļus sarežģītu ģeometrisko objektu aprakstam vai modelēšanai, no kā ir vajadzīgi renderēt lielu sarežģītu objektu no vairākiem primitīvas grupām.

OpenGL vidē pamat zīmēšanas objekti ir primitīvas (punkts, līnijas segments, daudzstūris, lauciņš) un tās zīmēšana tiek realizēta ar ēnotāju programmām un fiksētu funkciju apstrādes vienībām. Darbā tiek pieņemts izmantot punktus par objekta primitīviem un punkta primitīvus definē viena vai vairākas virsotņu grupas, kur viena virsotne ir primitīvas punkts vai galapunkts līnijā, vai daudzstūra stūris, kur satiekas divas malas utt. OpenGL ir iespējams zīmēt dažādas punkta primitīvas kombinācijas grupas/grupas secības (1.4 att.[[29]](#footnote-29)), bet darbā tiek paņemta klasiska trīsstūra secība (1.5 att.), jo trīsstūri ir vienīgie daudzstūri, kuras kombinācijas var veikt citas sarežģītas ģeometriskas formas, vienlaikus garantējot, ka tie atrodas arī vienā plaknē, kas nozīmē, ka tiem ir skaidri noteiktas un viegli aprēķināmas virsmas.

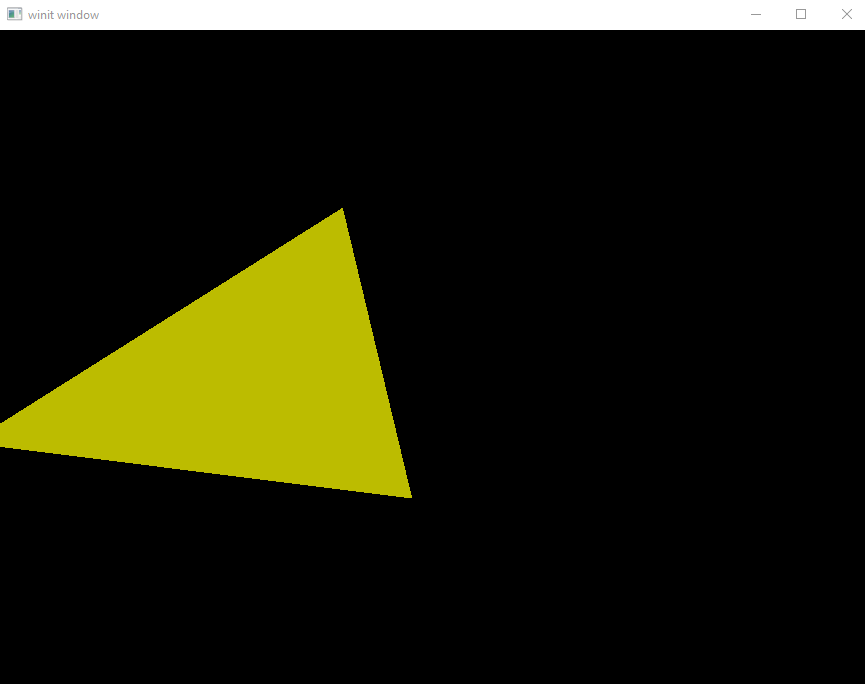


1.4 att. **OpenGL punkta primitīvas kombinācijas grupas/grupas secībās**



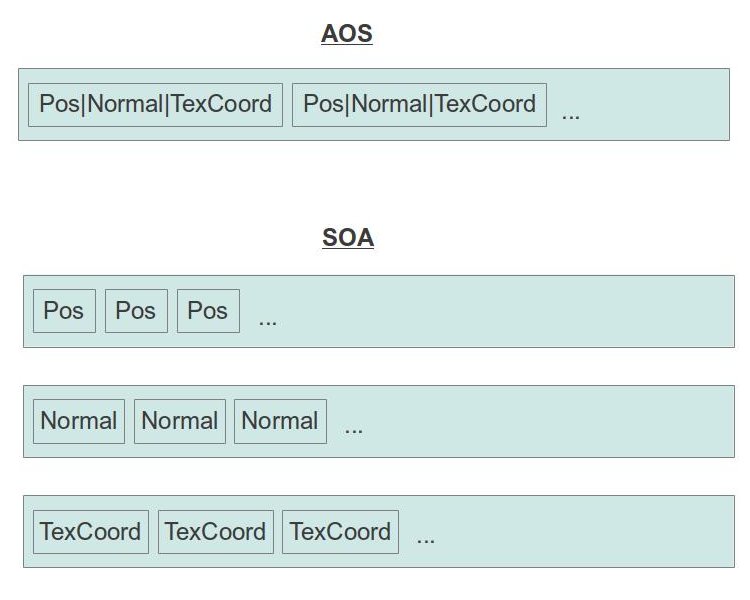
1.5 att. **OpenGL punkta primitīvas klasiska trīsstūra secība**

Katra virsotne tiek apstrādāta neatkarīgi no citām virsotnēm, un katrai virsotnei ir iespējams definēt tās pozīcijas koordinātas plaknē (pozīcijas vektors), krāsas (krāsas vektors), normālu (perpendikulārs punkta plaknei vektors), tekstūras koordinātas (tekstūras koordinātes vektors) utt. OpenGL logā (1.6 att.), kurus ir iespējams ierakstīt un saglabāt dažādos veidos, bet darbā būs paņemts viens no populārākiem datorgrafikas ģeometriska objekta saglabāšanas veidiem “.obj” faila formātā[[30]](#footnote-30).



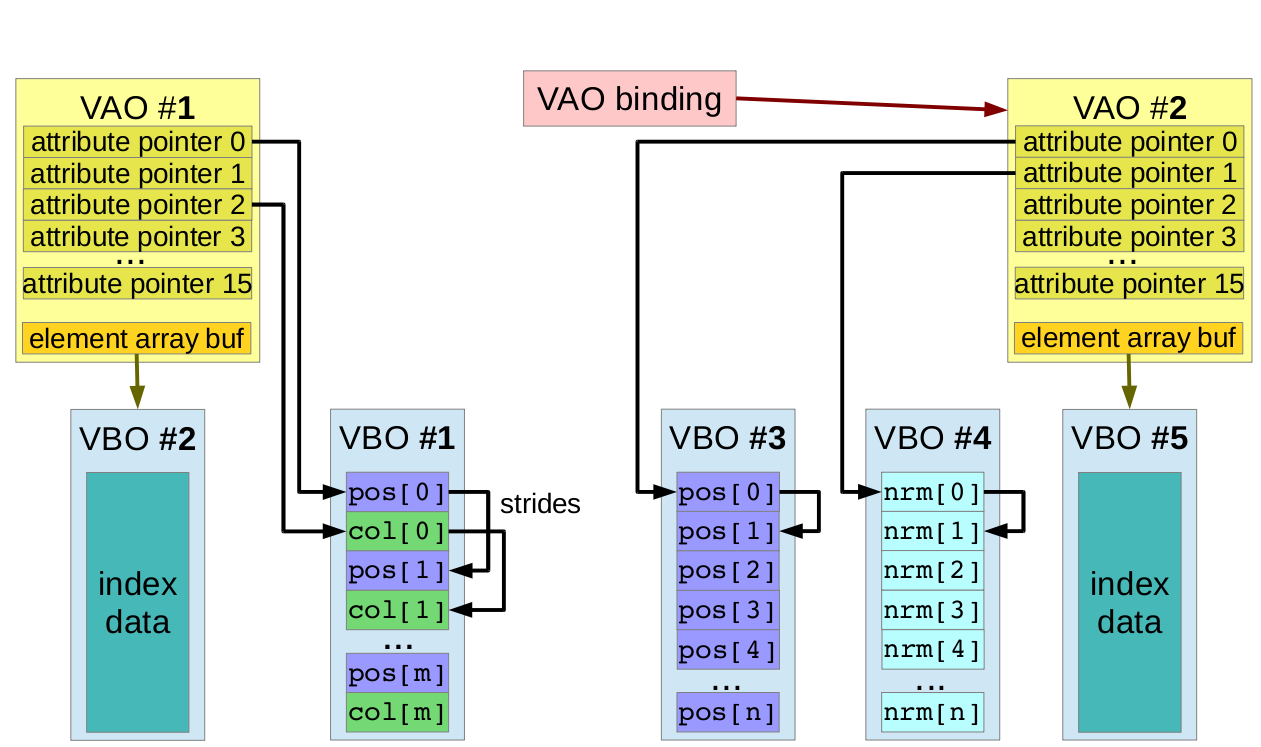
1.6 att. **OpenGL logs**

OpenGL vidē priekš katrām atsevišķam ģeometriska objekta virknes daļām, piemērām virsotnes virkne, virsotnes tekstūras virkne, virsotnes normālas vektora virkne, virsotnes grupēšanas secību utt., ir nepieciešams deklarēt virsotnes bufera objektu (VBO), kas ir atmiņas buferis videokartes atmiņā, kas paredzēts informācijas glabāšanai par virsotnēm, kurām nepieciešams deklarēt unikālo identifikatoru atmiņā (BufferID) un saistīt tās bufera datnes atmiņā. OpenGL vidē ir iespējams apvienot visas ģeometriska objekta daļas virknes vienā VBO, veikt struktūras masīvu (AOS), vai deklarēt atsevišķus VBO priekš katra ģeometriska daļa virkne, veikt masīvu struktūru (SOA), kā tiek paradīts piemērā (1.7 att.[[31]](#footnote-31)), kur objekta virsotnes (Pos), virsotnes normālas vektora (Normal), virsotnes tekstūras (TexCoords) virknes tiek saglabāti ar AOS un SOA metodem.



1.7 att. **OpenGL ģeometriska objekta daļas apvienošana metodes**

No tā, kā ģeometriska objekta VBO tiek deklarēti, bet OpenGL nevar definēt ģeometriska objekta virknes kopumu, ir svarīgi apvienot tos, un lai to izdarīt, OpenGL vidē eksistē virsotnes masīva objekts (VAO), kurš saglabā sevī katra deklarēta VBO izmēru, radītāju uz pirmo ierakstu, un katram VAO arī ir nepieciešams savs unikāls identifikators atmiņā (ArrayID), saistīt to atmiņā un rezultātā ģenerē jaunu VBO buferi ar indeksēs datnēs, kuru OpenGL izmanto ēnotāju programmās. Kā piemērs, attēlā (1.8 att.[[32]](#footnote-32)) ir paradīts AOS (kreisajā pusē) un SOA(labajā pusē) metodes ģeometriska objekta VBO apvienošana unikālā VAO.



1.8 att. **OpenGL VAO saistīšana ar VBO**

OpenGL komandu izpildes interpretācijas modelis ir bāzēts uz klients-serveris modeli, kur klients ir programma, kura izdod komandas, un serveris ir OpenGL, kurš apstrādā un interpretē komandas un pārveido uz dzelža līmenī. Visas komandas vienmēr tiek apstrādātas saņemšanas secībā, lai arī pirms komandas seku realizācijas var būt nenoteikts kavējums, piemērām, viens primitīvs ir jānozīmē pilnībā, pirms jebkurš nākamais var ietekmēt kadru buferi, vai arī vaicājumu un pikseļu lasīšanas operācijas atgriežas stāvoklī, kas atbilst visu iepriekš izsaukto OpenGL komandu pilnīgai izpildei, izņemot gadījumus, kad skaidri norādīts otrādāk.

### Globāls telpas attēla transformācijas

Lai veikt parastās datorgrafikas telpas transformācijas OpenGL vidē, piemērām telpas skatu punkta (kamera) pozīcija mainīšana, objektu starp x asi rotēšana, utt., ir pieņemts izmantot transformācijas matricas, kurās izmērs ir 4x4, jo OpenGL izmanto viendabīgas koordinātas (homogeneous coordinates) aprēķināšanās sistēma. No tā ir nepieciešams apskatīt darbā izmantojamas telpas transformācijas, kur tiek definētas modeļa (objekta), skatu (telpas kameras), projekcijas transformācijas matricas.

#### Modeļa transformācijas matrica

Modeļa transformācijas matrica tiek izmantota, lai aprakstīt un mainīt datorgrafikas objekta pozīciju, rotāciju, izmēru, un priekš katrai operācijai ir nepieciešamas savas izmaiņas objekta modeļa matricā, kura tiek reizināta ar virsotnes pozīcijas vektoru :

* Objekta virsotnes pozīcijas mainīšana ir aprakstīta ar formulu (1.1)

|  |  |  |
| --- | --- | --- |
|  | (1.1) | |
| , kur – x asi pārvietošanas vērtība;  – objekta virsotnes x asi pozīcijā;  – y asi pārvietošanas vērtība;  – objekta virsotnes y asi pozīcijā;  – z asi pārvietošanas vērtība;  – objekta virsotnes z asi pozīcijā; | |  |

* Objekta izvēru mainīšana ir aprakstīta ar formulu (1.2)

|  |  |  |
| --- | --- | --- |
|  | (1.2) | |
| , kur – x asi mērogošanas koeficients;  – objekta virsotnes x asi pozīcijā;  – y asi mērogošanas koeficients;  – objekta virsotnes y asi pozīcijā;  – z asi mērogošanas koeficients;  – objekta virsotnes z asi pozīcijā; | |  |

* Objekta rotācija tiek veikta apkārt vai , vai asi, un katrai rotācijai ir sava formula

1. Objekta rotācija apkārt X asi ir aprakstīta ar formulu (1.3)

|  |  |  |
| --- | --- | --- |
|  | (1.3) | |
| , kur – rotācijas leņķis;  – objekta virsotnes x asi pozīcijā;  – objekta virsotnes y asi pozīcijā;  – objekta virsotnes z asi pozīcijā; | |  |

1. Objekta rotācija apkārt Y asi ir aprakstīta ar formulu (1.4)

|  |  |  |
| --- | --- | --- |
|  | (1.4) | |
| , kur – rotācijas leņķis;  – objekta virsotnes x asi pozīcijā;  – objekta virsotnes y asi pozīcijā;  – objekta virsotnes z asi pozīcijā; | |  |

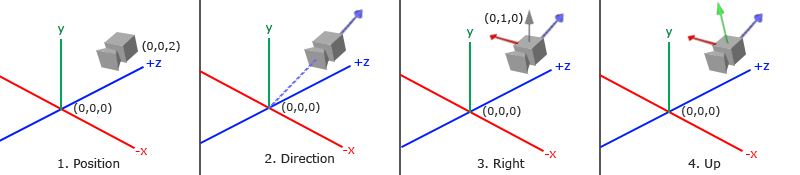
1. Objekta rotācija apkārt Z asi ir aprakstīta ar formulu (1.5)

|  |  |  |
| --- | --- | --- |
|  | (1.5) | |
| , kur – rotācijas leņķis;  – objekta virsotnes x asi pozīcijā;  – objekta virsotnes y asi pozīcijā;  – objekta virsotnes z asi pozīcijā; | |  |

Jā ir nepieciešams izveidot objekta vairākas transformācijas darbības, nepieciešams reizināt matricas viens ar otru un pēc tām reizināt ar pozīcijas vektoru.

#### Skatu transformācijas matrica

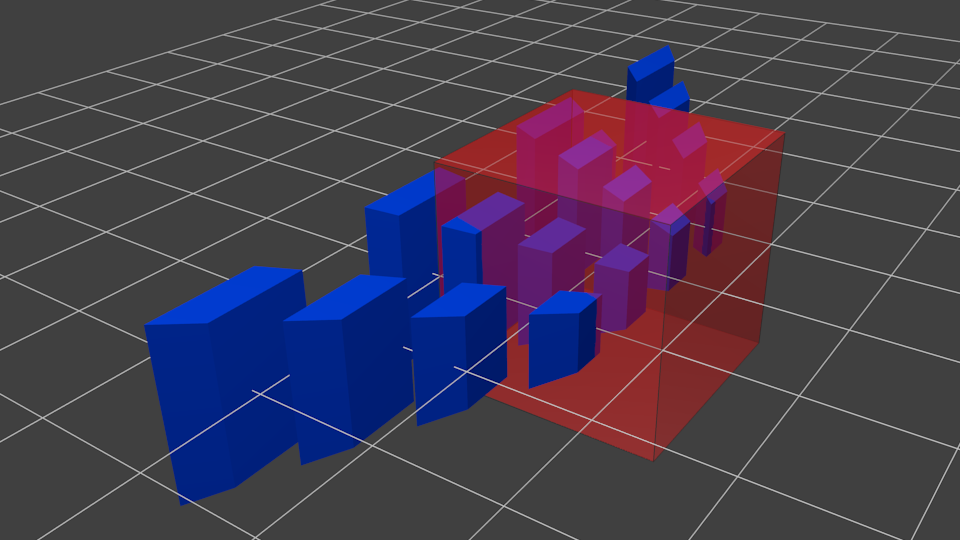
Skatu transformācijas matrica, vai telpas kamera matrica izgriež redzamo daļu no modelēta 3D pasaules koordinātes, kas ir saistītas ar kameras stāvokli un virzienu. Lai definēt kameru ir nepieciešams zināt tās pozīciju 3D pasaulē, skatu virzienu, kameras labas puses, augšpuses vektorus, kā tiek paradīts attēlā (1.9 att.[[33]](#footnote-33)), no labās puses uz kreisu.



1.9 att. **Kameras definēšanas parametri**

#### Projekcijas transformācijas matrica

No tā, kā datora monitors ir 2D virsma - ir nepieciešams, lai OpenGL 3D pasaule tiek projicēta uz 2D izskatu ar kopējo pasaules dziļumu un tiek attēlota OpenGL logā, ar to tiek jāizmanto pasaules projekcijas matricas transformāciju. Tas ir nepieciešams, lai konvertēt 3D pasaules koordinātes uz 2D normalizētas ierīces koordinātas (NDCs) ar globāla telpa dziļumu (1.10 att.[[34]](#footnote-34)), kur sarkans kubs ir NDC un zilie objekti - projektētās kuba objekti. Arī to izmanto, lai realizēt 3D pasaules objekta reālistisko apgaismojumu priekš katrā pikseļa fragmentu ēnotāju programmā, attālumu noteikšanai utt.



1.10 att. **Projekcijas matricas objektu transofrmācijas**

### Ēnotāju programmas jēdziens

Mūsdienas OpenGL renderēšanas process ir ļoti atkarīgs no ēnotāju izmantošanas, kā piemērs ir iespējams izpildīt tikai loga notīrīšanu neizmantojot ēnotājus, no tā ēnotāji ir vienas no svarīgākiem elementiem OpenGL vidē, ar kuriem ir iespējams veikt primitīvas matemātiskas pārveidošanas, apgaismošanas, teksturēšanas, ēnošanas operācijas utt.

OpenGL objekta grafiska apstrāde tiek sadalīta četros apstrādes posmos un viena aprēķināšanas posma, kurus kontrolē sava nodefinēta ēnotāja programma :

1. Virsotnes ēnojuma (The Vertex shading) posms – saņem virsotnes datus no virsotnes bufera objektos, apstrādājot katru virsotni atsevišķi. Šis posms ir obligāts visām OpenGL programmām, un tai jābūt saistītai ar ēnotāju.
2. Flīzēšanas ēnojuma (The Tessellation shading) posms – nav obligāts posms, kurā ir iespējams ģenerēt papildu ģeometriju priekš objekta virsotnēm.
3. Ģeometrijas ēnojuma (The Geometry shading) posms – nav obligāts posms, kurā ir iespējams modificēt veselus ģeometriskos primitīvus, ka piemērām mainīt ģeometriskās primitīvas veidu (piemēram, daudzstūrus pārveidojot par līnijām).
4. Fragmenta ēnojuma (The Fragment shading) posms - šajā posmā tiek apstrādāti atsevišķi fragmenti, kurus noģenerē OpenGL rasterizators un tiek aprēķinātas fragmenta krāsas un dziļuma vērtības, un pēc tam nosūtīts tālāk fragmenta pārbaudes apstrāde.
5. Aprēķināšanas ēnojuma (The Compute shading) posms – nav obligāts posms, kurā realizēts kadru bufera pēcapstrāde vai citas darbības ar renderētiem primitīviem, piemēram, statiskus objektus saglabāšana kešatmiņā.

Ēnotāju programmām ir sava sintakse un OpenGL izmanto GLSL valodu, kas ir līdzīgs C sintaksei, tabulā (1.11 tab.) lejā tiek paradītas ēnotāju programmas piemēri.

1.11 tabula

**Ēnotāju programmas kods un apraksts**

|  |  |
| --- | --- |
| Kods | Apraksts |
| #version 140  out vec4 color;  void main() {  color = vec4(0.5, 0.5, 0.0, 1.0);  } | Fragmenta ēnojuma programma, kurā tiek realizēta objekta pārzīmēšana dzeltenā krāsā. |
| #version 140  in vec2 position;  uniform mat4 matrix;  void main() {  gl\_Position = matrix \* vec4(position, 0.0, 1.0);  } | Virsotnes ēnojuma programma, kurā tiek realizēta objekta pārvietošana izmantojot transformācijas matricas reizinājums. |

OpenGL vidē ēnotāju programmas kopumu ir nepieciešams definēt ar identifikatoru (ProgramID) un saglabāt video kartes atmiņā, pirms ģeometriska objekta VAO deklarēšana, lai savienot ēnotāju programmas ienākošus parametrus (definētus ar “in” ēnotāju programmā) ar nepieciešamām datnēm, piemērām ģeometriska objekta VBO virsotnes vektorus, gaismas pozīcijas vektoru, tekstūras resursu, kuru arī ir nepieciešams saglabāt izmantojot OpenGL video kartēs atmiņā, identificēt (TextureID), utt.

### Objekta apgaismojums

Apgaismojums reālajā pasaulē ir ārkārtīgi sarežģīts un atkarīgs no pārāk daudziem faktoriem parādība, no tā OpenGL vidē apgaismojums tiek balstīts izmantojot vienkāršotus modeļus, kuri tiek balstītas uz gaismas fiziku teoriju. Viena no tādiem modeļiem ir “Phong” apgaismojuma modelis [4] (376. lpp.), kurā sastāv no 3 komponentiem (1.11 att.[[35]](#footnote-35)):

* **Apkārtējais apgaismojums** – apgaismojuma konstante, kurā vienmēr piešķir objektam krāsu, lai modelēt dabīgu situāciju, kā objekti dabā gandrīz nekad nav pilnīgi tumši.
* **Izkliedēts apgaismojums** –imitē gaismas objekta virziena ietekmi uz objektu. Šī ir vizuāli visnozīmīgākā apgaismojuma modeļa sastāvdaļa. Jo vairāk objekta daļa saskaras ar gaismas avotu, jo spilgtāka tā kļūst.
* **Spekulārs apgaismojums** – imitē gaismas spožo punktu, kas parādās uz spīdīgiem objektiem. Speciālie izceltie elementi ir vairāk tendēti uz gaismas krāsu, nevis uz objekta krāsu.



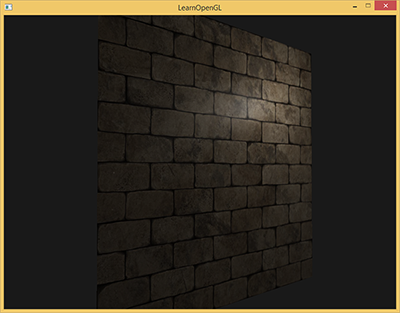
1.11 att. **“Phong” apgaismojuma modeļi**

Viss objekta apgaismojums balstīts uz objekta poligona plaknes normālu vektoriem, kā piemērs izkliedes apgaismojumam ir nepieciešams plaknes normālais vektors , lai noteikt plaknes apgaismojumu stiprumu no gaismas leņķa pret plaknei (1.12 att.[[36]](#footnote-36)).

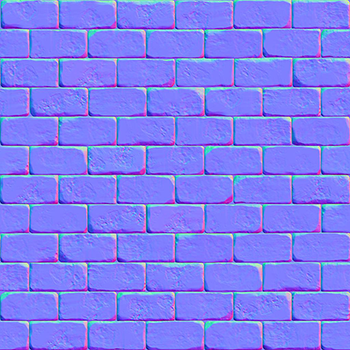


1.12 att. **Izkliedes apgaismojums OpenGL vidē**

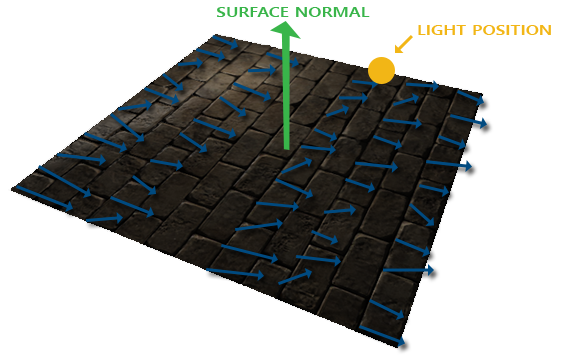
Bet jā izmantot apgaismojumu tikai pret virsmas normāliem vektoriem, tad teksturēti objekti zaudē savu dziļumu un izskatās plakni (1.13 att.[[37]](#footnote-37)), lai izlabot to ir pieņemts izmantot normāles tekstūras (1.14 att.[[38]](#footnote-38)), kurā zilā krāsa definē apgaismojuma vektora stiprumu asī, sarkana krāsa asī un zaļa asī, balstoties uz gaismas attālumu no objekta, kas nozīme, ka, jā skatīties uz objektu no katras pusēs, tad objekta apgaismojums tiek kļūst reālistisks, jo tiek pārdefinēta plaknes normāles vektori priekš laukumiem (1.15 att.[[39]](#footnote-39)).



1.13 att. **Objekta tekstūra bez normāles tekstūras**

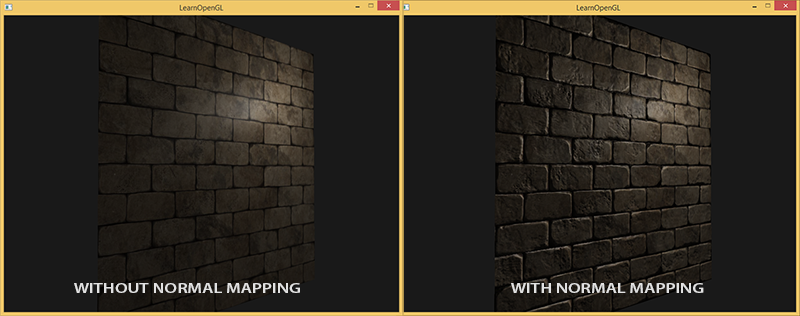


1.14 att. **Normāles vektoras tekstūra**



1.15 att. **Normāles tekstūras objekta apgaismojuma vektorus pārdefinēšana**

Rezultātā ir redzams, kā objekta apgaismojums kļūst reālistiskāk un pats objekts kļūst dziļāk (1.16 att.[[40]](#footnote-40)).



1.16 att. **Objekts bez un ar normāles tekstūras**

## Programmēšanas valodas salīdzināšanas plāns

Lai uzzinātu, kurai programmēšanas valodai ir priekšrocība datorgrafikā, ir jāizplāno programmēšanas valodu salīdzināšanas vispārēju struktūru, atrast rīkus, ar kuriem būs iespējams fiksēt eksperimentā laikā katras daudzobjektu renderēšanās programmas patērētas datora resursus, lai salīdzināt valodas resursu pārvaldības iespējās un noteikt priekšrocīgāko valodu katrā eksperimentā.

### Programmēšanas valodas vispārīga salīdzinājuma struktūra

Vispārīgam salīdzinājumam tiks izmantota lēmumu analizēšanas daudzkritēriju “Ideāla punktu” metode[[41]](#footnote-41) ar datnes normalizēšanu, kura ir aprakstīta ar formulu (1.6) un datnes normalizēšana maksimizēšanās gadījumā ar formulu (1.7), minimizēšanās gadījumā ar formulu (1.8), kurā tiek jādefinē alternatīvu kopumu un salīdzināšanas kritēriju kopumu. Tas ir nepieciešams, jo izmantojot “Ideāla punktu” metode ir iespējams nosaukt katras alternatīvas nosvērto attālumu no ideāla punkta, kas kopumā ļauj novērtēt visus alternatīvus, kur labāka ir aprakstīta ar formulu (1.9).

|  |  |
| --- | --- |
|  | (1.6) |
| , kur – i-tās alternatīvas nosvērtais attālums no ideāla punkta;  – i-tās alternatīvas un j-tā kritērija normalizētais vērtējums;  j-tā kritērija svars;  – kopējais kritēriju skaits; |  |

|  |  |
| --- | --- |
|  | (1.7) |
| , kur – i-tās alternatīvas un j-tā kritērija normalizētais vērtējums;  i-tās alternatīvas un j-tā kritērija vērtība;  j-tā kritērija maksimāla vērtība;  j-tā kritērija minimāla vērtība; |  |

|  |  |
| --- | --- |
|  | (1.8) |
| , kur – i-tās alternatīvas un j-tā kritērija normalizētais vērtējums;  i-tās alternatīvas un j-tā kritērija vērtība;  j-tā kritērija maksimāla vērtība;  j-tā kritērija minimāla vērtība; |  |

|  |  |
| --- | --- |
|  | (1.9) |
| , kur – i-tās alternatīvas nosvērtais attālums no ideāla punkta;  – kopējais alternatīvu skaits; |  |

No tā, kā “Ideāla punktu” metodē ir nepieciešams kritēriju svari, bija pieņemts izmantot “ranžēšanas”[[42]](#footnote-42) metode, jo izmantojot to ir iespējams ātri definēt katra kritērija vērtību pēc tās svarīgumā.

Kad ir nodefinēta salīdzināšanas metode ir svarīgi atzīmēt, kā vispārīgā salīdzinājumā C, Rust, GO un V programmēšanas valodas tiek izmantotas kā alternatīvas un kritēriji tiek definēti tabulā (1.12 tab.), kur valodas priekšrocības kopskaits jādefinē ar cita daudzkritēriju tabulu, kurā, kā alternatīvi arī ir C, Rust, GO un V programmēšanas valodas un kritēriji tiek definēti tabulā (1.13 tab.), jo daudzobjektu renderēšanas programmas resursu pārvaldības eksperiments būs veidots uz vairākiem datoriem.

1.12 tabula

**Valodas vispārīgā salīdzinājumā kritēriji**

|  |  |  |
| --- | --- | --- |
| **Kritērijs** | **Apraksts** | **↓min/↑max** |
| Kv1 – programmas simbolu skaits | Kopēja programmas projekta simbolu skaits. | ↓min |
| Kv2 – programmas uzticamība | Programmas kopēja datnes pārvaldes radītājs. | ↑max |
| Kv3 – programmas izstrādāšanas laiks | Patērēts laiks programmas izstrādāšanai. | ↓min |
| Kv4 – programmas izstrādāšanas centieni | Nepieciešamas darbības, lai sākt veidot OpenGL programmu. | ↓min |
| Kv5 – valodas priekšrocības kopskaits | Kopēja valodas priekšrocības kopskaits pamatojoties uz programmas efektivitātes eksperimentiem | ↑max |

1.13 tabula

**Programmas efektivitātes eksperimenta kritēriji**

|  |  |
| --- | --- |
| **Kritērijs** | **↓min/↑max** |
| Ke1 – programmas kompilācijas laiks | ↓min |
| Ke2 – programmas vidējais freimu (FPS) skaits | ↑max |
| Ke3 – programmas vidēja centrāla procesora (CPU) izmantošana | ↓min |
| Ke4 – programmas vidēja grafiska procesora (GPU) izmantošana | ↓min |
| Ke5 – programmas vidēja operatīvās atmiņas (RAM) izmantošana | ↓min |

Pamatojoties uz nodefinētām alternatīvu un kritēriju kopumam - ir nepieciešams veidot priekš katrā programmas efektivitātes eksperimenta tabulu, noteikt labāko alternatīvu katrā eksperimentā, pēc “Ideāla punktu” metode, sasummēt priekš katrā valoda kopējo skaitu, un izmantot to kā kritēriju vispārīgā salīdzinājuma tabulā, noteikt priekšrocīgāko valodas alternatīvu datorgrafikā, arī pēc “Ideāla punktu” metode, pieņemot to, kā priekš programmas efektivitātes eksperimenta tabulai kritēriju svari ir nodefinēti sekojoša tabulā (1.14 tab.), un priekš vispārīgā salīdzinājumā tabula ir nodefinēti sekojošā tabulā (1.15 tab.).

1.14 tabula

**Programmas efektivitātes eksperimenta kritēriju svari**

|  |  |  |
| --- | --- | --- |
|  | **Ke2 un Ke3 un Ke4 un Ke5** | **Ke1** |
| **Rangs** | 1 - 4 | 5 |
| **Svars** | 0.23 | 0.07 |

1.15 tabula

**Valodas vispārīgā salīdzinājumā kritēriju svari**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Kv5** | **Kv2** | **Kv4 un Kv3** | **Kv1** |
| **Rangs** | 1 | 2 | 3 - 4 | 5 |
| **Svars** | 0.33 | 0.27 | 0.17 | 0.07 |

### Programmas resursu pārvaldības eksperimenta datnes saņemšanas rīki

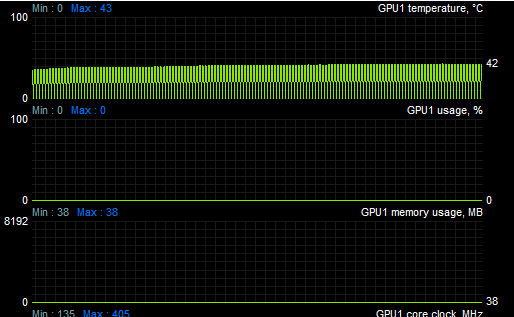
Lai nodefinēt katra valoda resursu pārvaldības kopējas iespējās - ir nepieciešams veidot vienādas renderēšanas programmas katrā valodā, kurā jārealizē liela datnes apjomu pārvaldību datorgrafikas kontekstā, un saņemt programmas izmantotas datora resursu vērtības strādāšanās laikā, no tā ir nepieciešams izmantot rīkus, kas palīdzē saglabāt katras programmas izmantotas datora resursu vērtības.

Pamatojoties uz to, ka datorgrafikas programmas vairāki datnes un resursu pārvaldības procesi, tiek realizēti izmantojot grafiska procesora resursus, bija pieņemts lēmums izmantot utilītprogrammu, ar kuru būtu iespējams, vispirms, fiksēt datora datorgrafikas procesa grafiska procesora kopējo resursu lietošanu un tikai pēc tām aplūkot tās centrāla procesora un operatīvas atmiņas lietošanas fiksēšanas iespējas. Aplūkojot mūsdienās labākas alternatīvas[[43]](#footnote-43), bija pieņemts izmantot “MSI Afterburner” ar “Rivatuner” utilītprogrammas.

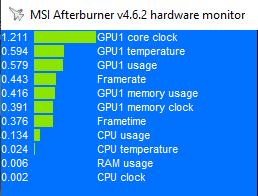
#### “MSI Afterburner” un “RivaTuner” rīki

“MSI Afterburner”[[44]](#footnote-44) - pasaulē atzītākā un plaši izmantotā grafisko procesora, centrāla procesora, kopēja operatīva atmiņa, utt. pārlūkošanas un datnes fiksēšanas utilītprogramma, kas sniedz pilnīgu kontroli pār kopējo resursu pārvalde datorgrafikas procesos, kā arī piedāvā izvēlēt un kombinēt vairākus pārlūkošanas parametrus, katrai datora komponentei.

Kopēja datnes pārlūkošana tiek realizēta ar grafikiem (1.10 att.), kurās tiek vizualizēta kopēja statistikas informācija priekš katra izvēlēta pārlūkošanas parametra, un histogrammām (1.11 att.), kurās tiek vizualizētas priekš katra izvēlēta pārlūkošanas parametra pēdējā izmaiņas vērtība, bet arī ir iespējams redzēt kopējo “MSI Afterburner” datora resursu informāciju par datorgrafikas programmu, datorgrafikas programmas logā, izmantojot iekļauto programmas izpildīšanas laikā monitoringa rīku “RivaTuner”[[45]](#footnote-45), ar kura palīdzību tiek realizēta ērta kopēja datorgrafikas programmas apkopota informācija no “MSI Afterburner”.



1.17 att. **”MSI Afterburner” grafiskā veidā datora resursu pārlukošanas piemērs**



1.18 att. **“MSI Afterburner” histogrammas veidā datora resursu pārlukošanas piemērs**

Izmantojot “MSI Afterbuner” ir iespējams ne tikai aplūkot programmas procesa resursu lietošanu, bet arī fiksēt tos failā laikā periodā, priekš katra datorgrafikas procesa, kas kopumā ļauj saņemt aktuālākas datorgrafikas programmas resursu lietošanas datnes “.hml” vai “.csv” formātā, kur failā tiek ierakstīti visi nedefinēti “MSI Afterburner” pārlūkošanas parametri laikā vienībā.

Ar šiem rīkiem ir iespējams ātrāk saņemt nepieciešamas datnes priekš daudzobjektu renderēšanas programmām, bet “MSI Afterburner” rīkā nav iespējams fiksēt tieši datorgrafikas programmas izmantoto operatīva atmiņu, tikai kopējo sistēmu operatīva atmiņu patēriņu, no tā bija pieņemts rakstīt savu “PowerShell” skriptu, kurš palaiž nodefinēto programmu, tiek fiksēts un ierakstīts failā palaista programmas procesa patērēta operatīva atmiņa laikā vienībā (MB mērvienībā), līdz programmas beigās, jo nebija atrasts rīks, ar kuru būtu iespējams fiksēt viena programma operatīva atmiņa patēriņu.

# C, go, rust, V opengl renderēšanas progRammas algoritmi un valodas salīdzināšana

Šajā nodaļā tiek aprakstīti viena un daudzobjektu programmas algoritmi, daudzobjektu programmas efektivitātes salīdzināšanas piemērs un kopēja vispārīgā valodas salīdzināšanā, pēc kurās rezultāta tiek nodefinēta vai ir mūsdienīga alternatīva C valodai datorgrafikā, pamatojoties uz OpenGL programmas rezultātiem.

## Viena un daudzobjektu renderēšanas programmas algoritmi

Pirms programmas izstrādāšanai ir nepieciešams nodefinēt kopējo programmas algoritmu, kurā tiek paradīta un aprakstīta kopēja programmas ideja, soļu secībā tekstu veidā un ar algoritma shēmu.

, un arī programmas realizācijās procesā noskaidrot nepieciešamie izstrādāšanas centieni, lai katrā valodā varētu realizēt viena renderēšanas objekta programmu.

### Viena objekta renderēšanas programmas algoritms

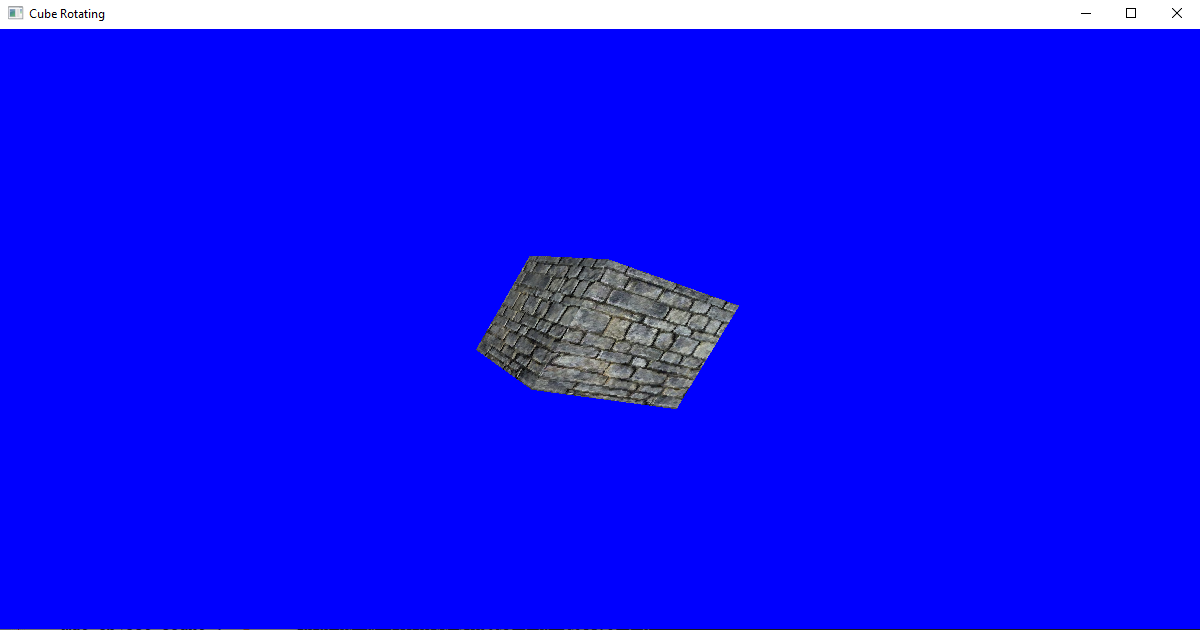
Lai sāktu salīdzināt katrā valodā uzrakstīto programmu resursu patēriņu, ir jāīsteno vienkāršas sākotnējās viena teksturēta objekta ar gaismas atspīdumu renderēšanas programmas, uz kuru pamata tiks balstītas daudzobjektu renderēšanas programmas.

Ar OpenGL ir dažādas iespējas, kā realizēt viena objekta renderēšanas algoritmu, bet bija realizēts tāds algoritms, kurā tiek veikta transformācija tikai ar ģeometriska objekta modeļa matricu, kas nozīme, ka kameras un projekcijas transformācijas matricas, ēnotāju programma, tekstūras, ģeometriska objekta VAO un VBO netiek pārdefinēta katra freimā laikā.

Algoritms tiek definēts ar sekojošiem soļiem (algoritma shēmu ir iespējams aplūkot 2.pielīkumā):

1. Programmas sākums;
2. Pārbaudīt vai eksistē OpenGL savienošanu ar sistēmu, jā eksistē, tad pāriet 3. solī, jā neeksistē, tad pāriet 16. solī;
3. Saistīt OpenGL sistēmā;
4. Atvērt OpenGL logu;
5. Ielādēt ēnotāju programmu kopumu un saglabāt identifikatoru;
6. Ielādēt tekstūras un saglabāt tās identifikatorus;
7. Ielādēt un saglabāt objekta virsotnes kopumu;
8. Aprēķināt kameras un projekcijas matricas;
9. Saistīt objekta VAO un VBO;
10. Pārbaudīt vai ir ciets OpenGL logs, jā nav ciets, tad pāriet 11. solī, jā ciets pāriet 15. solī;
11. Notīrīt OpenGL logu;
12. Aprēķināt objekta jaunu modeļa transformācijas matricu;
13. Savienot VAO, VBO, tekstūras identifikatorus (TextureID), kameras, projekcijas, objekta modeļa transformācijas matricas ar ēnotāju programmas kopumu;
14. uzzīmēt VAO un pāriet 10. solī;
15. Atbrīvot video kartes atmiņu no VAO, VBO, tekstūrām un ēnotāju programmām;
16. Pabeigt programmu;

Pēc algoritma realizācijas, bija izveidota viena teksturēta objekta ar gaismas atspīdumu renderēšanas programma katrā valodā (2.1 att.).



2.1 att. **Viena teksturēta objekta ar gaismas atspīdumu renderēšanas programma V valodā**

### Daudzobjektu renderēšanas programmas algoritms

Lai izveidot daudzobjektu renderēšanas programmas katrā valodā, bija nepieciešams uzlabot viena objekta renderēšanas programmas algoritmu, pēc kurā tiek izstrādātas galīgas salīdzināšanas programmas priekš katra valoda.

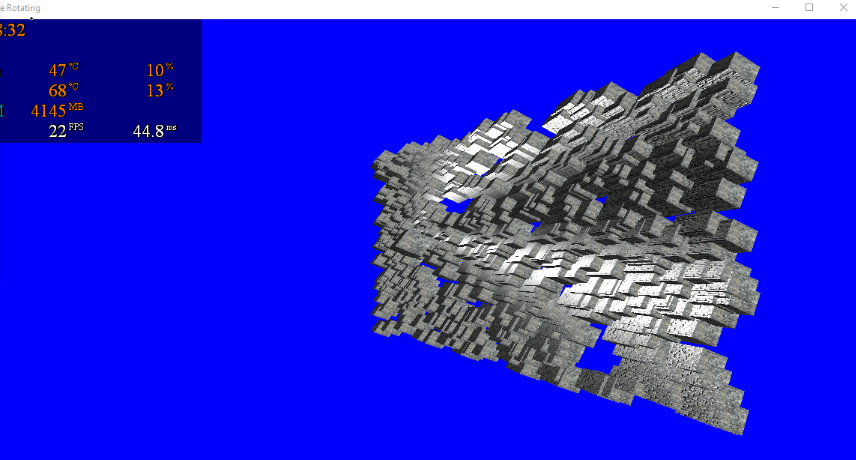
Priekš daudzobjektu renderēšanas programmām bija izmainīts objekta renderēšanas pamatprincips - objekta VAO un VBO tika saistīti un dzēsti katrā renderēšanasposmā, lai neaizņemt grafiska procesora un operatīvas atmiņu ar nevajadzīgām datnes buferiem.

Šīs renderēšanas pamatprincips arī palīdz apskatīt, kādā valodas programmā ir labāka atmiņas pārvaldība un kopēja resursa pārvaldības ātrums, kad ir nepieciešams strādāt ar vairākiem dažādiem objektiem un tās atmiņas buferiem.

Algoritms tiek definēts ar sekojošiem soļiem (algoritma shēmu ir iespējams aplūkot 3.pielīkumā):

1. Programmas sākums;
2. Pārbaudīt vai eksistē OpenGL savienošanu ar sistēmu, jā eksistē, tad pāriet 3. solī, jā neeksistē, tad pāriet 27. solī;
3. Saistīt OpenGL sistēmā;
4. Atvērt OpenGL logu;
5. Ielādēt ēnotāju programmu kopumu un saglabāt identifikatoru;
6. Ielādēt tekstūras un saglabāt tās identifikatorus;
7. Izveidot objektu sarakstu;
8. Ielādēt no “.obj” faila objektu un saglabāt objekta virsotnes kopumu;
9. Pievienot jaunu objektu sarakstā;
10. Aprēķināt kameras un projekcijas matricas;
11. Pārbaudīt vai ir ciets OpenGL logs, jā nav ciets, tad pāriet 12. solī, jā ciets pāriet 26. solī;
12. Jā pagāja viena sekunde, pāriet 13. soli, jā nē, tad 15. soli;
13. Ielādēt no “.obj” faila objektu un saglabāt objekta virsotnes kopumu;
14. Pievienot jaunu objektu sarakstā;
15. Jā objektu skaits ir lielāks nekā 500, tad pāriet 26. solī, jā nē, tad pāriet 16. solī;
16. Definēt i = 0;
17. Notīrīt OpenGL logu;
18. Jā i vērtība ir lielāk nekā saraksta garums, tad pāriet 11. solī, jā nē, tad pāriet 19. solī.
19. Paņemt no saraksta i-to objektu;
20. Aprēķināt objekta jaunu modeļa transformācijas matricu;
21. Saistīt objekta VAO un VBO;
22. Savienot VAO, VBO, tekstūras identifikatorus (TextureID), kameras, projekcijas, objekta modeļa transformācijas matricas, gaismas vektoru ar ēnotāju programmas kopumu;
23. Uzzīmēt VAO;
24. Atbrīvot video kartes atmiņu no i-tā objekta VAO, VBO;
25. Pievienot 1 pie i un pāriet 18. solī;
26. Atbrīvot video kartes atmiņu no tekstūrām un ēnotāju programmām;
27. Pabeigt programmu;

Pamatojoties uz algoritma programmas realizāciju, katrā valodā bija izveidota daudzobjektu renderēšanās programmas (2.2 att.), ar kurām tiek veikti galīgie valodas salīdzinājumi.



2.2 att. **Daudzobjektu renderēšanās programma V valodā.**

## C, GO, Rust, V salīdzināšana

Balstoties uz daudzobjektu renderēšanas programmām rezultātiem ir iespējams nodefinēt vai ir C programmēšanas valodas alternatīva datorgrafikā, kurai ir priekšroka OpenGL vidē.

Lai pilnvērtīgi salīdzināt katru programmu un uzzināt kurai valodai ir priekšroka vispārīgā salīdzinājumā, nepieciešams saņemt programmas efektivitātes eksperimenta datnes vismaz no pieciem dažādiem datoriem (2.1 tab.), pēc tām nodefinēt programmēšanas centienus, un kopējo valodas uzticamību.

2.1 tabula

|  |  |  |  |
| --- | --- | --- | --- |
| **Kārtas numurs** | **Centrāla procesora modelis** | **Grafiska procesora modelis** | **Operatīvas atmiņas daudzums** |
| 1. | Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz | NVIDIA GeForce GTX 960M (2GB) | 8 GB |
| 2. | Intel(R) Core(TM) i5-7600K CPU @ 3.80GHz | NVIDIA GeForce RTX 2070 (8GB) | 8 GB |
| 3. | Intel(R) Core(TM) i3-8100 CPU @ 3.60GHz | NVIDIA GeForce GTX 650 (4GB) | 8 GB |
| 4. | Intel(R) Core(TM) i3-8100 CPU @ 3.60GHz | NVIDIA GeForce GTX 1070 (8GB) | 16 GB |
| 5. | Intel(R) Core(TM) i3-8100 CPU @ 3.60GHz | AMD Radeon RX 580 (8GB) | 12 GB |

### Programmas efektivitātes eksperiments

Katrā eksperimentā priekš katrā valodā tiek nodefinēts, ka maksimālais objektu skaits ir 500 objekti, kā tiek minēts daudzobjektu renderēšanās algoritma aprakstā. Katras programmas datnes mērīšana sadalīta uz “PowerShell” skripta un “MSI Afterburner” datu ieguve, kur “PowerShell” skripts palaiž programmu un saglāba katrā sekunde programmas paņemto operatīvas atmiņu un “MSI Afterburner” saglabā centrāla procesora, grafiska procesora procentuālo paņemto resursu daudzumu, un freimu daudzumu sekundē, kas parāda cik ātri tiek renderēts objektu kopums.

Priekš piemēra ir paradīts un secināts pirmā datora eksperimentu rezultāti:

* Operatīvas atmiņas grafisks rezultāts:
* Grafiska procesora izmantošanas rezultāts:
* Kadru ātrums rezultāts:
* Centrāla procesora izmantošanas rezultāts:

Rezultāta bija saņemtas vidēji rezultāti katrai valodai (2.2 tab.), kurus nepieciešams pārveidot normalizētā veidā un ar “Ideāla punkta” metode noteikt priekšrocīgāko valodu pirmā datora eksperimentā (2.3 tab.). Arī bija pamanīts, kā V valodā paradījās atmiņas noplūde, kura nebija saistīta ar programmas realizēšanas defektu, bet ar kopējo V valodas lielas datnes sarakstam pārvaldības problēmām.

2.2 tabula

**Pirmā datora vidējas vērtības rezultāti**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Valoda** | **Kompilācijas laiks (sek.)** | **Kadru ātrums (freims/sek.)** | **Centrāla procesora izmantošana (%)** | **Grafiska procesora izmantošana (%)** | **Operatīva atmiņa (MB)** |
| **C** | 2.12 | 223.41 | 24.34 | 32.49 | 44.22 |
| **V** | 2.09 | 211.29 | 24.14 | 32.56 | 242.77 |
| **GO** | 2.43 | 196.10 | 19.46 | 36.56 | 46.48 |
| **Rust** | 5.21 | 62.87 | 19.27 | 37.87 | 33.93 |

2.3 tabula

**Pirmā datora eksperimenta rezultāti pēc “Ideāla punkta” metode ar normalizētām vērtībām**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Ke1** | **Ke2** | **Ke3** | **Ke4** | **Ke5** |  |
| **Svars** | **0.07** | **0.23** | **0.23** | **0.23** | **0.23** |  |
| **C** | 0.99 | 1 | 0 | 1 | 0.95 | 0.24 |
| **V** | 1 | 0.93 | 0.04 | 0.99 | 0 | 0.47 |
| **GO** | 0.89 | 0.83 | 0.96 | 0.24 | 0.94 | 0.25 |
| **Rust** | 0 | 0 | 1 | 0 | 1 | 0.53 |
|  | ↓ min | ↑ max | ↓ min | ↓ min | ↓ min |  |

Pēc pirmā datora eksperimenta rezultātiem ir redzams, kā priekšrocīgāka valoda ir C, GO valoda atpaliek tikai par 0.01 punktu, bet Rust ar V atpaliek gandrīz par divām reizēm.

Visas citas datorus rezultātus ir iespējams aplūkot 4. pielikumā.

### Programmēšanas centieni

Katrai valodai ir nepieciešamas vairākas darbības, lai saistīt to ar OpenGL un sākt izmantot to katrā datorā, no tā bija izveidota tabula (), kurā paradītas nepieciešamas darbības lai saistīt katru valodu ar OpenGL.

|  |  |  |  |
| --- | --- | --- | --- |
| **C** | **Rust** | **GO** | **V** |
| MinGW kompilatorā instālētas GL OpenGL bibliotēka ar x64 bitu versiju | Instālēt GLIUM pakete projektā | Instālēt GL pakete projektā | Noradīt V kompilatoram C valodas MinGW kompilatoru |
| MinGW kompilatorā instālētas GLFW OpenGL bibliotēka ar x64 bitu versiju |  | Instālēt mgl32 pakete projektā | MinGW kompilātorā jābūt instalētas GL, GLFW, GLM bibliotēkās |
| MinGW kompilatorā instālētas GLM OpenGL bibliotēka ar x64 bitu versiju |  | Instālēt GLFW pakete projektā | Instālēt freetype bibliotēkas |
| Bibliotēkas pievienošana projektā ar Cmake |  |  | Priekš x64 versijām papildināt V kodu priekš pamat V OpenGL bibliotēkām savienošanās ar sistēmu |

# Parasts 3d redaktors

SECINĀJUMI

Apkopojot

LITERATŪRA

1. Chris Woodford, *Computer Graphics,* 10. septembrī 2018. gads [skatīts 14. decembrī 2019. gadā] Pieejams: <https://www.explainthatstuff.com/computer-graphics.html>

2. Autora nav, *OpenGL vs. DirectX: A Comparison,* [skatīts 14.decembrī 2019. gadā] Pieejams: https://www.cprogramming.com/tutorial/openglvsdirectx.html

3. Dmitry Guzeev, *What is OpenGL and how to start learning it?,* 19. janvāris 2018. gads [skatīts 14. decembrī 2019. gadā] Pieejams: <https://medium.com/@dmitrygz/what-is-opengl-and-how-to-start-learning-it-34f19cfa219f>

4. Dave Shreiner, Graham Sellers, John Kessenich, Bill Licea-Kane, *OpenGL Programming Guide Eighth Edition*, Michigan, 2013. gads, 986 lpp Pieejams: https://www.cs.utexas.edu/users/fussell/courses/cs354/handouts/Addison.Wesley.OpenGL.Programming.Guide.8th.Edition.Mar.2013.ISBN.0321773039.pdf

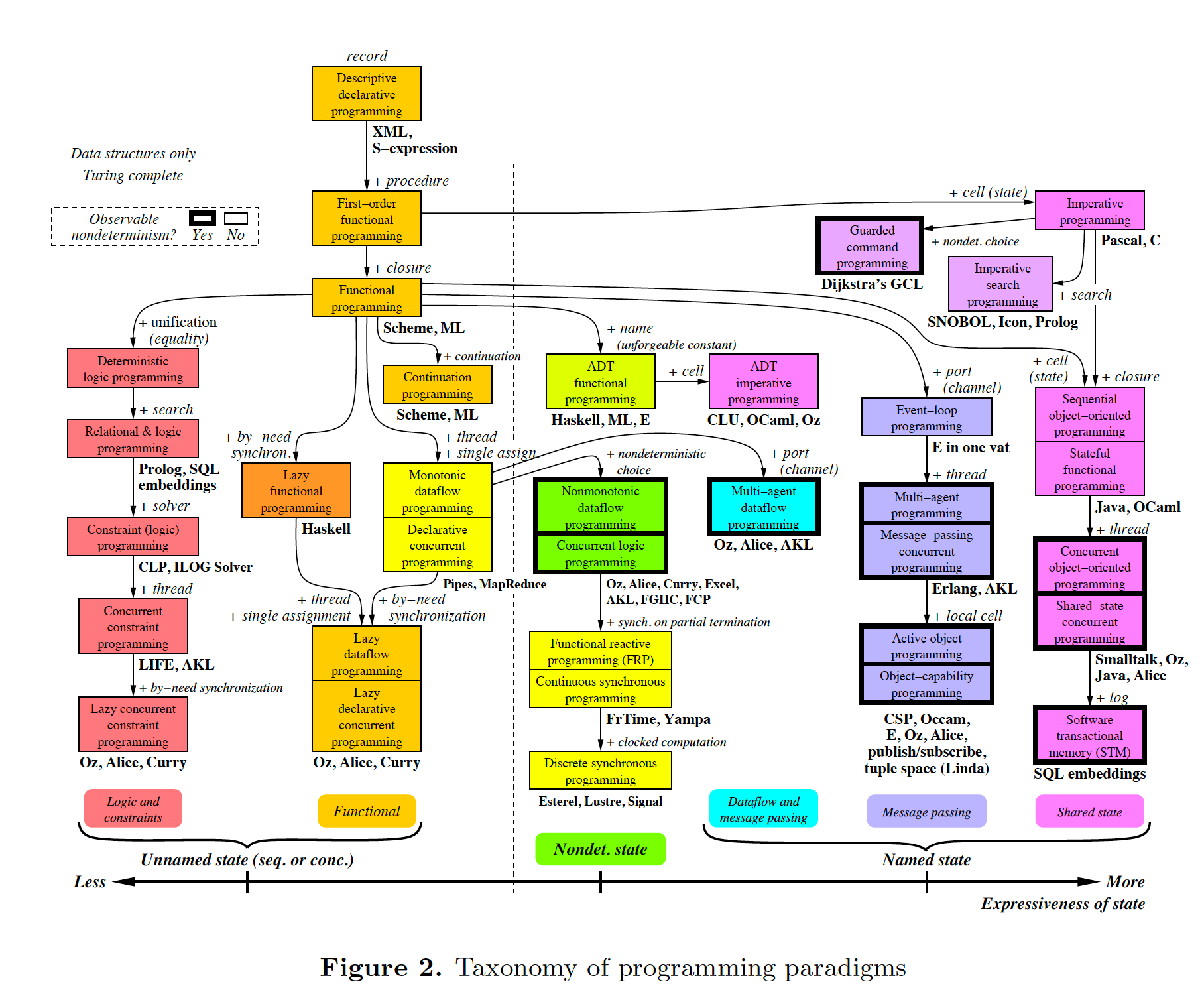
5. [Andreas Zwinkau](https://beza1e1.tuxen.de/index.html), *Faster Than C*, 24. februāris 2013. gads [skatīts 14. decembrī 2019. gadā], Pieejams: <https://beza1e1.tuxen.de/articles/faster_than_C.html>

6. [Andrew Meredith](https://dev.to/kendru)*, Can Rust replace C/C++ as system language?,* 12. septembris 2019. gads [skatīts 14. decembrī 2019. gadā], Pieejams: <https://dev.to/kendru/can-rust-replace-c-c-as-a-systems-language-7c7>

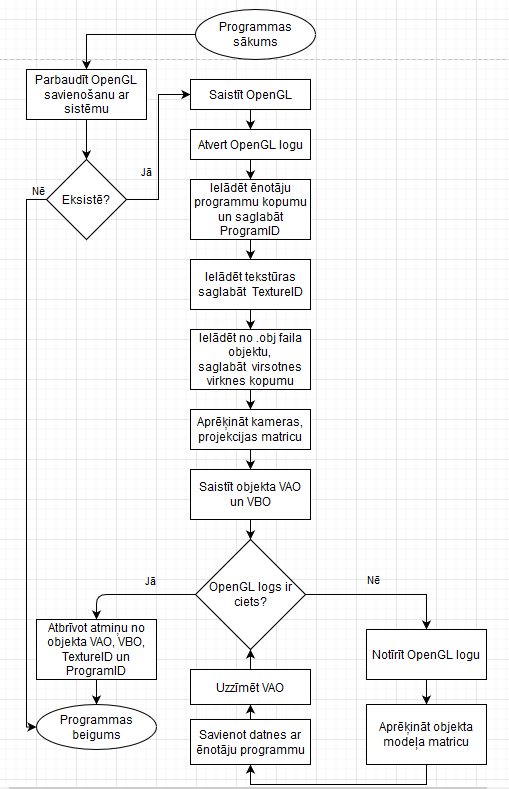
1. Professor Marsette (Marty) Vona, *Computer Graphics Suggested Platforms*, 21. decembris 2019. gads [skatīts 21. decembrī 2019. gadā] Pieejams: http://www.ccs.neu.edu/home/fell/CS5310/platforms.html
2. Alexander Medvednikov, *Comparison of V and Other Languages,* februāris 2019. gads [skatīts 11. decembrī 2019. gadā], Pieejams: <https://vlang.io/compare>
3. Steve Klabnik un Carol Nichols, *The RUST programming language,* 2018. gads, 617. lpp. Pieejams: <https://lise-henry.github.io/books/trpl2.pdf>
4. Miek Gieben, *Learning Go,* 2012. gads 112. lpp. Pieejams: <https://miek.nl/files/go/Learning-Go-latest.pdf>
5. Alexander Medvednikov, *The V Programming Language*, februāris 2019. gads [skatīts 9. februārī 2020. gadā], Pieejams: <https://vlang.io/>
6. Mark Segal, Kurt Akeley, Chris Frazier, Jon Leech, Pat Brown, *OpenGL 4.6 Core Profile*, Michigan, 22. oktobris 2019. gads, 850 lpp Pieejams: https://www.khronos.org/registry/OpenGL/specs/gl/glspec46.core.pdf
7. Alexander Medvednikov, *The V Programming Language Documentation*, februāris 2019. gads [skatīts 9. februārī 2020. gadā], Pieejams: <https://vlang.io/docs>

PIELIKUMI

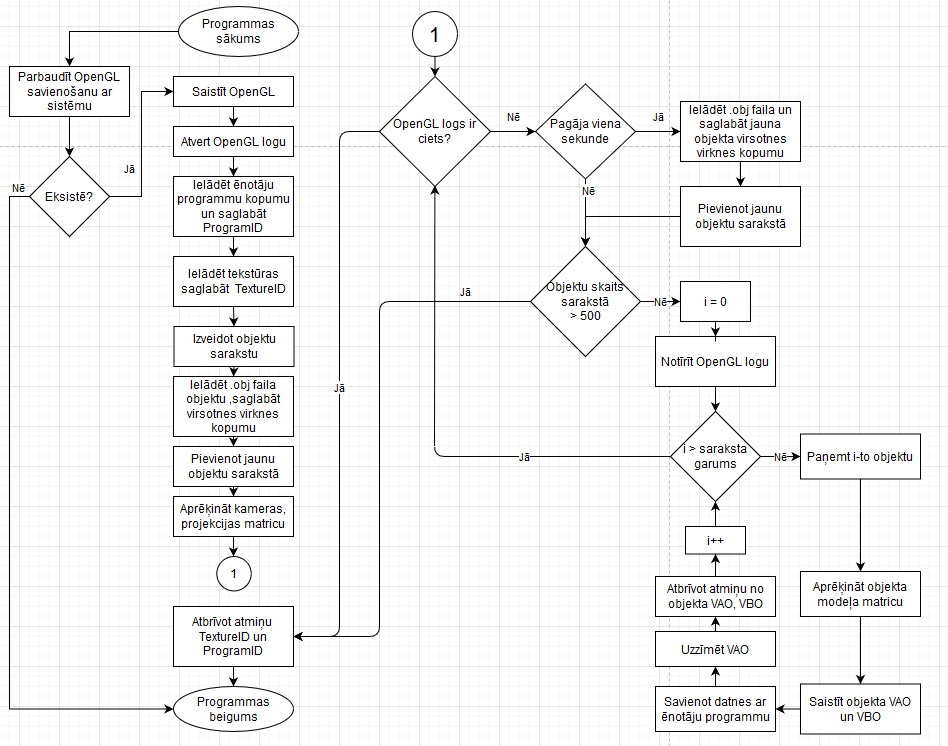
1. pielikums  
   Paradigmas taksonomijas attēls[[46]](#footnote-46)



1. pielikums Viena objekta renderēšanas programmas algoritms



1. pielikums Daudzobjektu renderēšanas programmas algoritms



1. pielikums Eksperimenta rezultāti

**Otrā datora vidējas vērtības rezultāti**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Valoda** | **Kompilācijas laiks (sek.)** | **Kadru ātrums (freims/sek.)** | **Centrāla procesora izmantošana (%)** | **Grafiska procesora izmantošana (%)** | **Operatīva atmiņa (MB)** |
| **C** | 0.90 | 583.78 | 24.58 | 13.70 | 45.37 |
| **V** | 0.89 | 516.06 | 23.28 | 12.47 | 1631.87 |
| **GO** | 1.02 | 128.89 | 22.88 | 11.79 | 60.10 |
| **Rust** | 0.93 | 133.38 | 24.01 | 7.89 | 31.40 |

**Otrā datora eksperimenta rezultāti pēc “Ideāla punkta” metode ar normalizētām vērtībām**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Ke1** | **Ke2** | **Ke3** | **Ke4** | **Ke5** |  |
| **Svars** | **0.07** | **0.23** | **0.23** | **0.23** | **0.23** |  |
| **C** | 0.92 | 1.00 | 0 | 0 | 0.99 | 0.47 |
| **V** | 1 | 0.85 | 0.76 | 0.21 | 0 | 0.50 |
| **GO** | 0 | 0.00 | 1 | 0.33 | 0.98 | 0.46 |
| **Rust** | 0.69 | 0.01 | 0.33 | 1 | 1 | 0.40 |
|  | ↓ min | ↑ max | ↓ min | ↓ min | ↓ min |  |

**Trešā datora vidējas vērtības rezultāti**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Valoda** | **Kompilācijas laiks (sek.)** | **Kadru ātrums (freims/sek.)** | **Centrāla procesora izmantošana (%)** | **Grafiska procesora izmantošana (%)** | **Operatīva atmiņa (MB)** |
| **C** | 0.66 | 472.44 | 28.63 | 36.83 | 43.45 |
| **V** | 0.75 | 502.50 | 29.27 | 32.13 | 227.52 |
| **GO** | 0.81 | 436.68 | 29.68 | 38.83 | 42.08 |
| **Rust** | 0.69 | 116.55 | 31.21 | 24.28 | 31.39 |

**Trešā datora eksperimenta rezultāti pēc “Ideāla punkta” metode ar normalizētām vērtībām**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Ke1** | **Ke2** | **Ke3** | **Ke4** | **Ke5** |  |
| **Svars** | **0.07** | **0.23** | **0.23** | **0.23** | **0.23** |  |
| **C** | 1 | 0.92 | 1.00 | 0.14 | 0.94 | 0.23 |
| **V** | 0.4 | 1.00 | 0.76 | 0.21 | 0.00 | 0.51 |
| **GO** | 0 | 0.83 | 1.00 | 0.33 | 0.98 | 0.27 |
| **Rust** | 0.8 | 0.00 | 0.33 | 1.00 | 1.00 | 0.40 |
|  | ↓ min | ↑ max | ↓ min | ↓ min | ↓ min |  |

**Ceturtā datora vidējas vērtības rezultāti**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Valoda** | **Kompilācijas laiks (sek.)** | **Kadru ātrums (freims/sek.)** | **Centrāla procesora izmantošana (%)** | **Grafiska procesora izmantošana (%)** | **Operatīva atmiņa (MB)** |
| **C** | 0.89 | 545.97 | 31.45 | 28.88 | 35.84 |
| **V** | 0.92 | 476.85 | 31.96 | 26.75 | 243.92 |
| **GO** | 0.96 | 390.42 | 34.27 | 23.88 | 41.77 |
| **Rust** | 0.94 | 115.95 | 28.63 | 17.91 | 28.08 |

**Ceturtā datora eksperimenta rezultāti pēc “Ideāla punkta” metode ar normalizētām vērtībām**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Ke1** | **Ke2** | **Ke3** | **Ke4** | **Ke5** |  |
| **Svars** | **0.07** | **0.23** | **0.23** | **0.23** | **0.23** |  |
| **C** | 1.00 | 1.00 | 0.50 | 0.00 | 0.96 | 0.35 |
| **V** | 0.57 | 0.84 | 0.41 | 0.19 | 0.00 | 0.62 |
| **GO** | 0.00 | 0.64 | 0.00 | 0.46 | 0.94 | 0.52 |
| **Rust** | 0.29 | 0.00 | 1.00 | 1.00 | 1.00 | 0.28 |
|  | ↓ min | ↑ max | ↓ min | ↓ min | ↓ min |  |

**Piektā datora vidējas vērtības rezultāti**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Valoda** | **Kompilācijas laiks (sek.)** | **Kadru ātrums (freims/sek.)** | **Centrāla procesora izmantošana (%)** | **Grafiska procesora izmantošana (%)** | **Operatīva atmiņa (MB)** |
| **C** | 0.64 | 343.12 | 28.43 | 45.39 | 87.63 |
| **V** | 0.71 | 266.82 | 28.47 | 49.41 | 300.42 |
| **GO** | 0.85 | 320.92 | 34.26 | 42.46 | 98.53 |
| **Rust** | 0.62 | 113.21 | 27.39 | 21.42 | 41.14 |

**Piektā datora eksperimenta rezultāti pēc “Ideāla punkta” metode ar normalizētām vērtībām**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Ke1** | **Ke2** | **Ke3** | **Ke4** | **Ke5** |  |
| **Svars** | **0.07** | **0.23** | **0.23** | **0.23** | **0.23** |  |
| **C** | 0.91 | 1.00 | 0.85 | 0.14 | 0.82 | 0.28 |
| **V** | 0.61 | 0.67 | 0.84 | 0.00 | 0.00 | 0.60 |
| **GO** | 0.00 | 0.90 | 0.00 | 0.25 | 0.78 | 0.55 |
| **Rust** | 1.00 | 0.00 | 1.00 | 1.00 | 1.00 | 0.23 |
|  | ↓ min | ↑ max | ↓ min | ↓ min | ↓ min |  |

Es, Aleksandrs Koroļko, bakalaura studiju programmas „Informācijas tehnoloģija” III RDBI01 grupas students(-e), ar savu parakstu apstiprinu, ka esmu izstrādājis(-usi) doto bakalaura darbu, kas iesniegts Rīgas Tehniskajā universitātē inženierzinātņu bakalaura grāda datorvadībā un datorzinātnē iegūšanai.

Bakalaura darbs ir izpildīts pilnīgi patstāvīgi un satur visas nepieciešamās atsauces uz darbā izmantotajiem materiāliem.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(studenta paraksts)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Bakalaura darbs izstrādāts Vadības informācijas tehnoloģijas katedrā

Darba autors: Aleksandrs Koroļko \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(paraksts)

201\_. gada \_\_\_. \_\_\_\_\_\_\_\_\_\_\_

Zinātniskais vadītājs: Artūrs Braučs \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (paraksts)

201\_. gada \_\_\_. \_\_\_\_\_\_\_\_\_\_\_

Bakalaura darbs pielaists aizstāvēšanai:

Vadības informācijas tehnoloģijas katedras vadītājs: Jānis Grabis \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(paraksts)

201\_. gada \_\_\_. \_\_\_\_\_\_\_\_\_\_\_

Bakalaura darbs aizstāvēts Informācijas tehnoloģijas institūta bakalaura darbu aizstāvēšanas komisijas 201\_. gada \_\_\_.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ sēdē, protokola Nr. \_\_\_\_\_\_\_\_ un novērtēts ar atzīmi \_\_\_\_\_\_\_\_\_.

Informācijas tehnoloģijas institūta bakalaura darbu aizstāvēšanas komisijas sekretāre: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /doc.V.Minkēviča/

1. Stack Overflow aptauja par populārākajiem programmēšanas valodām / Ir pieejams – [https://insights.stackoverflow.com/survey/2019#most-popular-technologies](https://insights.stackoverflow.com/survey/2019%23most-popular-technologies) [↑](#footnote-ref-1)
2. Tīmekļa raksts “10 top Programming Languages in 2020 for businesses” / autors [Ruchika Singh Aggarwal](https://codeburst.io/@ruchikaggarwal89?source=post_page-----a2921798d652----------------------) / datums 12. jūnijs 2020 / Ir pieejams - <https://codeburst.io/10-top-programming-languages-in-2019-for-developers-a2921798d652> [↑](#footnote-ref-2)
3. Tīmekļa raksts “After All These Years, the World is Still Powered by C Programming” / Ir pieejams - <https://www.toptal.com/c/after-all-these-years-the-world-is-still-powered-by-c-programming> [↑](#footnote-ref-3)
4. Tīmekļa raksts ““Why C is Popular” / Ir pieejams - <https://data-flair.training/blogs/why-c-is-popular/> [↑](#footnote-ref-4)
5. Tīmekļa raksts “Why C is so popular and still the best programming language” / Ir pieejams - <https://www.geekboots.com/story/why-c-is-so-popular-and-still-the-best-programming-language> [↑](#footnote-ref-5)
6. Tīmekļa raksts “Importance of code quality” / autors Nishitha Rufus / datums 4. februāris 2019. gads / Ir pieejams - <https://www.infognana.com/importance-of-code-quality/> [↑](#footnote-ref-6)
7. Tīmekļa raksts “Why speed is critical in Software Development” / datums 30. aprīlis 2019. gads / Ir pieejams - <https://online.wlu.ca/news/2019/04/30/why-speed-critical-software-development> [↑](#footnote-ref-7)
8. Līdzīgas produkcijas apskatīšana un salīdzināšana tīmekļa veidne / Ir pieejams [www.slant.co](http://www.slant.co) [↑](#footnote-ref-8)
9. Tīmekļa raksts “What exactly is a programming paradigm?” / autors M.V.Thanoshan / datums [12. novembris 2019. gads] / Ir pieejams <https://www.freecodecamp.org/news/what-exactly-is-a-programming-paradigm/> [↑](#footnote-ref-9)
10. Tīmekļa raksts “” / autors Bhumika Rani/ Ir pieejams <https://www.geeksforgeeks.org/introduction-of-programming-paradigms/> [↑](#footnote-ref-10)
11. Tīmekļa raksts “Programming paradigms” / Ir pieejams <https://cs.lmu.edu/~ray/notes/paradigms/> [↑](#footnote-ref-11)
12. Tīmekļa IBM raksts par Rust / autors M. Tim Jones / datums [8. marts 2018. gadā] / ir pieejams <https://developer.ibm.com/technologies/web-development/articles/os-developers-know-rust/> [↑](#footnote-ref-12)
13. Attēla ir pieejama <https://developer.ibm.com/developer/articles/os-developers-know-rust/images/Figure01.png> [↑](#footnote-ref-13)
14. Rust kompilators “rustc” / Ir pieejams <https://doc.rust-lang.org/rustc/what-is-rustc.html> [↑](#footnote-ref-14)
15. Rust paketes menedžeris “Cargo” / Ir pieejams <https://doc.rust-lang.org/cargo/index.html> [↑](#footnote-ref-15)
16. Tīmekļa raksts “Rust for JavaScript Developers” / autors David Morcillo / datums [15. oktobris 2019. gads] / Ir pieejams <https://www.codegram.com/blog/rust-for-js-developers/> [↑](#footnote-ref-16)
17. Tīmekļa raksts “Getting Started With Rust, the friendly System Programming Language” / autors Dhruvil Trivedi / datums [21. jūlijs 2018. gads] / Ir pieejams <https://opensourceforu.com/2018/07/getting-started-with-rust-the-friendly-systems-programming-language/> [↑](#footnote-ref-17)
18. Attēla ir pieejama <https://i2.wp.com/opensourceforu.com/wp-content/uploads/2018/07/Figure-2-Graphical-comparison-of-Rust.jpg?ssl=1> [↑](#footnote-ref-18)
19. Rust FAQ par lēnu kompilāciju / Ir pieejams [https://prev.rust-lang.org/en-US/faq.html#why-is-rustc-slow](https://prev.rust-lang.org/en-US/faq.html%23why-is-rustc-slow) [↑](#footnote-ref-19)
20. Tīmekļa raksts “Rust – memory safety without garbage collector” / autors Yan Cui / datums [5. aprīlis 2017. gads] / Ir pieejams - <https://medium.com/theburningmonk-com/rust-memory-safety-without-garbage-collector-d6a25b23c036> [↑](#footnote-ref-20)
21. Tīmekļa atskaite par GO Lang programmēšanas valodu / Ir pieejams - <https://kuree.gitbooks.io/the-go-programming-language-report/content/16/text.html> [↑](#footnote-ref-21)
22. Tīmekļa raksts “Why should you learn GO?” / autors Kavel Patel / datums [8. janvārī 2017. gads] / Ir pieejams - <https://medium.com/@kevalpatel2106/why-should-you-learn-go-f607681fad65> [↑](#footnote-ref-22)
23. GO GC kompilators / Ir pieejams <https://golang.org/doc/faq#What_compiler_technology_is_used_to_build_the_compilers> [↑](#footnote-ref-23)
24. Apmācības GO Lang veidnē / Ir pieejams <https://www.golang-book.com/books/intro/10> [↑](#footnote-ref-24)
25. Oficiāla FAQ atbilde uz GO Lang mērķi / Ir pieejams [https://golang.org/doc/faq#What\_is\_the\_purpose\_of\_the\_project](https://golang.org/doc/faq%23What_is_the_purpose_of_the_project) [↑](#footnote-ref-25)
26. Vienlaicīgumā un koda caurspīdīgumā valodas grafika attēls / Ir pieejama [https://miro.medium.com/max/994/1\*xbsHBQJReC5l\_VO4XgNSIQ.png](https://miro.medium.com/max/994/1*xbsHBQJReC5l_VO4XgNSIQ.png) [↑](#footnote-ref-26)
27. Pilna C Lang nepieciešama vieta ar kompilatoru - <https://lists.llvm.org/pipermail/llvm-dev/2019-April/132028.html> [↑](#footnote-ref-27)
28. Pilna Swift nepieciešama vieta ar kompilatoru - [https://github.com/apple/swift#getting-started](https://github.com/apple/swift%23getting-started) [↑](#footnote-ref-28)
29. OpenGL primitīvas kombinācijas grupas/grupas secības attēls / Ir pieejams - <https://www3.ntu.edu.sg/home/ehchua/programming/opengl/images/GL_GeometricPrimitives.png> [↑](#footnote-ref-29)
30. Faila formāta “.obj” apraksta lapa / autors Paul Bourke / Ir pieejams – <http://paulbourke.net/dataformats/obj/> [↑](#footnote-ref-30)
31. OpenGL virsotnes masīva objekta apvienošanas metodes attēls / Ir pieejams <http://ogldev.org/www/tutorial32/aos_soa.jpg> [↑](#footnote-ref-31)
32. OpenGL VAO saistīšana ar VBO attēls / Ir pieejams <https://i.stack.imgur.com/nk4L0.png> [↑](#footnote-ref-32)
33. Kameras definēšanas parametri attēls / Ir pieejams [https://learnopengl.com/img/getting-started/camera\_axes.png](https://learnopengl.com/img/getting-started/camera_axes.png%20) [↑](#footnote-ref-33)
34. Projekcijas matricas transformācijas attēls / Ir pieejams - <http://www.opengl-tutorial.org/assets/images/tuto-picking-physics-library/homogeneous.png> [↑](#footnote-ref-34)
35. “Phong” apgaismojuma modeļa attēls / Ir pieejams - <https://learnopengl.com/img/lighting/basic_lighting_phong.png> [↑](#footnote-ref-35)
36. Izkliedes apgaismojuma attēls / Ir pieejams - <https://learnopengl.com/img/lighting/diffuse_light.png> [↑](#footnote-ref-36)
37. Objekta tekstūra bez normāles tekstūras attēls / Ir pieejams - <https://learnopengl.com/img/advanced-lighting/normal_mapping_flat.png> [↑](#footnote-ref-37)
38. Normāles vektora tekstūra attēls / Ir pieejams - [https://learnopengl.com/img/advanced-lighting/normal\_mapping\_normal\_map.png](https://learnopengl.com/img/advanced-lighting/normal_mapping_normal_map.png%20) [↑](#footnote-ref-38)
39. Normāles tekstūras objekta apgaismojuma vektorus pārdefinēšana attēls / Ir pieejams - <https://learnopengl.com/img/advanced-lighting/normal_mapping_ground_normals.png> [↑](#footnote-ref-39)
40. Objekts bez un ar normāles tekstūras attēls / Ir pieejams - <https://learnopengl.com/img/advanced-lighting/normal_mapping_compare.png> [↑](#footnote-ref-40)
41. Lēmumu analīzes metodes “Daudzkriteriju uzdevumi : Risināšanas metodes” prezentācija (94. slaids) / autors RTU DITF ITI MIK profesore Ludmila Aleksejeva / Ir pieejams - [https://estudijas.rtu.lv/pluginfile.php/1706120/mod\_resource/content/0/- B11\_L\_2019-20\_Daudzkriteriju uzdevumu risinasanas metodes.pdf](https://estudijas.rtu.lv/pluginfile.php/1706120/mod_resource/content/0/-%20B11_L_2019-20_Daudzkriteriju%20uzdevumu%20risinasanas%20metodes.pdf) [↑](#footnote-ref-41)
42. Lēmumu analīzes metodes “Subjektīvo mērījumu metodes” prezentācija (10. slaids) / autors RTU DITF ITI MIK profesore Ludmila Aleksejeva / Ir pieejams - [https://estudijas.rtu.lv/pluginfile.php/1700413/mod\_resource/content/0/- B09\_L\_2019-20\_Subjektivas vertesanas metodes.pdf](https://estudijas.rtu.lv/pluginfile.php/1700413/mod_resource/content/0/-%20B09_L_2019-20_Subjektivas%20vertesanas%20metodes.pdf) [↑](#footnote-ref-42)
43. Datora resursu pārvaldības fiksēšanas labākas utilītprogrammas “17 best overclocking sofware (CPU/GPU/RAM) – Free&Paid - 2020” saraksts / Ir pieejams - <https://www.tech21century.com/best-free-overclocking-software-cpu-gpu-ram/> [↑](#footnote-ref-43)
44. “MSI Afterburner” utilītprogrammas timekļa lapa / Ir pieejams - <https://www.msi.com/page/afterburner> [↑](#footnote-ref-44)
45. “RivaTuner” diagnostika un reālā laika monitoringa utilītprogramma / Ir pieejams - <https://www.guru3d.com/content-page/rivatuner.html> [↑](#footnote-ref-45)
46. Ir pieejams <https://medium.com/@jingchenjc2019/a-brief-survey-of-programming-paradigms-207543a84e2b> [↑](#footnote-ref-46)