**МЕЃУНАРОДЕН СЛАВЈАНСКИ УНИВЕРЗИТЕТ**
**СВЕТИ НИКОЛЕ**
**РЕПУБЛИКА СЕВЕРНА МАКЕДОНИЈА**




**ФАКУЛТЕТ ЗА ТЕХНИЧКИ НАУКИ И ИНФОРМАТИКА**
**СТУДИСКА ПРОГРАМА: СОФТВЕРСКИ ТЕХНОЛОГИИ**

МАГИСТЕРСКИ ТРУД

# ПОЧЕТНИ ФАЗИ НА КРЕАЦИЈА НА ГРАФИЧКИ КОРИСНИЧКИ ИНТЕРФЕЈС

**СТУДЕНТ:** ТОМИСЛАВ БУНДАЛЕВСКИ
**БРОЈ НА ИНДЕКС:** M229005
**МЕНТОР:** ПРОФ. Д-Р МИМОЗА. А. ЈАНКУЛОВСКА

**Свети Николе, 2024**

*На семејството*

# Апстракт

UI framework претставува апстракција во која се обезбедува генеричка функционалност и може да се менува со дополнителен код. Еден UI framework обезбедува стандарден начин за градење и распоредување на GUI апликации и е софтверска средина за долгорочна употреба која обезбедува посебна функционалност како дел од поголема софтверска платформа за олеснување на развојот на софтверски апликации.

Windows обезбедува многу примитивни алатки за градење кориснички интерфејси. Системот обезбедува неколку основни контроли и природни контејнери за прозорци, но изградбата на сопствен кориснички интерфејс е тешка.

Корисничкиот интерфејс (UI) во овој труд е воден според дизајн документите на софтверот Chromium и нивниот прилагоден views UI framework за прикажување на “chrome” UI делот на прелистувачот.

Целта на овој труд беше да се демонстрира еден од многуте начини на кој би можеле да почнеме да градиме свој UI framework, независно од самиот оперативен систем, графичко API или слично, со што се исполни генералната хипотеза и задачата на ова истражување.



**Клучни зборови:** framework, GUI, Windows, Chromium, Skia

# Abstract

A UI framework is an abstraction that provides generic functionality and can be modified with additional code. A UI framework is a standard way of building and deploying GUI applications and is a long-term software environment that contains functionality from an open source software platform.

Windows provides many primitive tools for building user interfaces. The system provides several basic controls and native window containers, but building a custom own user interface is difficult.

The user interface (UI) in this paper is guided by the design documents of the Chromium software and their custom views UI framework for displaying the "chrome" UI part of the browser.

The purpose of this paper was to demonstrate one of the many ways in which we could start building our own UI framework, independent of the operating system itself, graphic API or similar, thus fulfilling the general hypothesis and task of this research.



**Keywords:** framework, GUI, Windows, Chromium, Skia

# Содржина


Апстракт.....3

Листа на слики.....7

Користени кратенки.....10

Вовед.....12

1. Софтверски framework (рамка).....14
2. Апликациски framework (рамка).....16
   - Историја.....16
3. Компјутерска графика.....17
   - 3.1. Видови на слики.....18
     - Дводимензионални.....18
     - Пикселна уметност.....19
     - Спрајт графика.....19
     - Векторска графика.....19
     - Генеративни модели за машинско учење.....20
     - Тридимензионални.....21
     - Компјутерска анимација.....21
   - 3.2. Концепти и принципи.....23
     - Пиксел.....23
     - Основни елементи (примитивни) .....23
     - Рендерирање.....24
   - 3.3. Шејдер (shader) .....24
     - Уредувачи на шејдери преку GUI.....27
4. Дизајн на кориснички интерфејс.....28
   - Во споредба со UX дизајн.....29
5. Графички кориснички интерфејс (GUI) .....29
   - 5.1. GUI и дизајн на интеракција.....30
     - Компоненти.....31
   - 5.2. GUI дрво.....33
6. Графичка библиотека или графичко API.....34
   - OpenGL.....34
   - Vulkan.....36
   - Metal.....37
   - GDI/GDI+ (Graphics Device Interface) .....38
   - DirectX (Direct3D) .....40
   - ANGLE.....45

- Cairo……46
- Skia.....48
- Управување со боја на Skia…….....49
- Оптимизации…….……..……50
- Класи и објекти на Skia…….……..…50
- SkSL и ефекти за време на извршување……..52
- Skia во другите софтвери како Chromium…….....53
- Користење на Skia…….....56

7. Методологија на истражувањето.....58
   - Цели на истражувањето.....58
   - Задачи на истражувањето.....58
   - Генерална хипотеза.....59
   - Методи на истражувањето.....59
   - Оправданост на истражувањето.....59
8. Практичен дел.....60

Заклучок.....104

Користена литература и линкови.....106

# Листа на слики

Слика 1. Растерска и наспроти векторска слика кога ќе се претстави на поголема резолуција……..……..……..……...20
Слика 2. Пример за компјутерска анимација произведена со “Motion Capture”……...22
Слика 3. Линеарна интерполација……..……..……..…22
Слика 4. Пример за два вида шејдери: рамно засенчување лево и Phong засенчување надесно……..……..…….……..……...…25
Слика 5. Друга употреба на шејдерите е за специјални ефекти, дури и на 2Д слики, (на пр., фотографија од веб-камера). Непроменетата, незасенчена слика е лево, а истата слика има шејдер применет десно……..……..……..…..26
Слика 6. Уредувачот на шејдери во Unreal Engine 4. Интерфејсот на уредувачот на шејдери се состои од лента со мени, лента со алатки и четири отворени панели…….28
Слика 7. GUI е претставен (прикажан) на екранот на компјутерот. Резултат на обработен влез на корисникот и главниот интерфејс за интеракција човек-машина……..……31
Слика 8. Слоеви на GUI базирани на систем за прозорци……..…….33
Слика 9. GUI дрво……..……..…….……...……33
Слика 10. Илустрација на процесот на OpenGL графичкиот pipeline……..……..36
Слика 11. OpenGL и Vulkan се API за рендерирање. Во двата случаи, графичкиот процесор извршува шејдери, додека процесорот извршува сè друго……..…….....37
Слика 12. GDI во архитектурата на Windows NT……..……..…..…40
Слика 13. Апстрактен слој помеѓу GDI и Direct3D……..……41
Слика 14. Direct3D уред, неговите ресурси и синџири за размена (Swap Chains).….....42
Слика 15. Direct3D 11 процес на графички цевковод, односно pipeline……..……..…44
Слика 16. Моделот на цртање во Каиро……..……..……47
Слика 17. Пример во C/C++ како се користи Cairo. Долу десно е резултатот генериран од самиот код……..……..……..……...……47
Слика 18. Пример во C/C++ како се користи Skia. Долу е резултатот генериран од самиот код, соодветно од страна на CPU и GPU……..……..……...…56
Слика 19. Пример во C/C++ како се користи Skia. Долу е резултатот генериран од самиот код, соодветно од страна на CPU и GPU……..……..……...…57

Слика 20. Креирање на проект во Visual Studio и конфигурирање на C++ верзија и компајлер……..……..……..……60
Слика 21. Поставување на Clang/LLVM во Windows System Environment variables……..……..……..……...…61
Слика 22. Компајлирање на Skia со ANGLE backend……..……....63
Слика 23. Резултатот од компајлирање на Skia со ANGLE backend……..…….63
Слика 24. Поставување на Skia include директориуми……..……..64
Слика 25. Поставување на Skia и Windows константи……..……...65
Слика 26. Поставување на Runtime Library во /MT……..……..…..66
Слика 27. Поставување на Additional Library Directories……..……...…67
Слика 28. Поставување на Additional Dependencies……..……...…68
Слика 29. Копирање на потребните фајлови. Истите фајлови може да се најдат во другите познати софтвери како и прелистувачите……..……..…….69
Слика 30. Тестирање на Skia……..……..……...70
Слика 31. Делумна декларација на фонт менаџер “FontMgr” ……..……..…71
Слика 32. Делумна имплементација на фонт менаџер “FontMgr” ……..……...…71
Слика 33. Преправање во точна синтакса на кодот за вчитување на фонт и цртање на текст……..……..……..…….72
Слика 34. Тестирање на Skia……..……..……...73
Слика 35. Chromium Skia команди генерирани од едно HTML дрво со CSS својства (правоаголници)……..……..……..……74
Слика 36. Истите Skia команди извршени од проектот (правоаголници).……..……...…75
Слика 37. Chromium Skia команди генерирани од едно HTML дрво со CSS својства (заоблени правоаголници).……..……..……..75
Слика 38. Истите Skia команди извршени од проектот (заоблени правоаголници).……....76
Слика 39. Цртање на слика преку Skia……..……..……...…77
Слика 40. Цртање на слика со ефекти преку Skia……..……..…….77
Слика 41. Креирање на апликација со 2 прозорци……..……..…79
Слика 42. Windows Message Loop……..……..……...…79
Слика 43. Иницијализација на ANGLE со Skia……..……..……..80

Слика 44. Регистрирање на класа, креирање на класа и справување со настани за еден прозорец……..……..……..……..80
Слика 45. Имплементација на прозорец……..……..……81
Слика 46. Прозорци коишто рендерираат со графичката карта користејќи ANGLE со Skia……81
Слика 47. Изворниот код за цртање на border, outline и box-shadow. Во конкретниот пример, се користи само border……..……....82
Слика 48. Демонстрација на CSS border, со сите 4 различни страни……..……...…83
Слика 49. Демонстрација на CSS outline……..……..……...…83
Слика 50. Демонстрација на CSS box-shadows……..……..…….84
Слика 51. Демонстрација на изворниот код за цртање на CSS border. Специфично, ова е почетокот, каде што се одлучува по кој логичен пат треба да се нацрта тој border…….85
Слика 52. Демонстрација на изворниот код за цртање на CSS outline. Специфично, ова е почетокот, каде што се одлучува по кој логичен пат треба да се нацрта тој outline…….85
Слика 53. Демонстрација на изворниот код за цртање на CSS box-shadows……..…...…86
Слика 54. Демонстрација на цртање на текст како прелистувачот Chromium……..…87
Слика 55. Демонстрација на цртање на текст како прелистувачот Chromium, користејќи десно-кон-лево писмо……..……..…….…88
Слика 56. Демонстрација на Flexbox layout користејќи ја библиотеката Yoga и цртање на позициите и големините пресметани од Yoga преку Skia. Горната слика претставува output од прелистувач користејќи HTML и CSS, додека долниот прозорец е наша имплементација на CSS својства преку C++……..……..…….93
Слика 57. Демонстрација на изминување на едно дрво за цртање……..……...……94
Слика 58. Демонстрација на “hit-testing” на едно дрво……..…….…….98
Слика 59. Демонстрација на чекање на сигналот за “VSync” со 165Hz……..……...…..101
Слика 60. Демонстрација на чекање на сигналот за “VSync” со 24Hz……..……..…….102
Слика 61. Демонстрација на користење SkCubicMap, крива која може да се користи за да се постигне анимација на екран……..……..……..……..103

## Користени кратенки

| Име на кратенка | Description (English) | Опис (Македонски) |
|---|---|---|
| API | Application Programming Interface | Програмски интерфејс за апликации |
| SDK | Software Development Kit | Комплет за развој на софтвер |
| UI | User Interface | Кориснички интерфејс |
| GUI | Graphical User Interface | Графички кориснички интерфејс |
| SkSL | Skia Shading Language | Јазик за засенчување на Skia |
| OpenGL | Open Graphics Library | - |
| ANGLE | Almost Native Graphics Layer Engine | - |
| HLSL | High-Level Shader Language | - |
| GLSL | OpenGL Shading Language | - |
| MSL | Metal Shading Language | - |
| HTML | HyperText Markup Language | - |
| CSS | Cascading Style Sheets | - |
| VSync | Vertical Synchronization | Вертикална синхронизација |
| CPU | Centralized Processing Unit | Централизирана единица за обработка (процесор) |
| GPU | Graphical Processing Unit | Единица за графичка обработка (графичка) |
| OOP | Object-Oriented Programming | Објектно-ориентирано програмирање |
| CGI | Computer Generated Image | Слика генерирана од компјутер |
| PNG | Portable Network Graphics | - |

| | | |
|---|---|---|
| JPG | Joint Photographic Expert Group | - |
| SVG | Scalable Vector Graphics | - |
| PDF | Portable Document Format | - |
| GDI | Graphics Device Interface | - |
| LLVM | Low Level Virtual Machine | - |
| HWND | Handle to a Window | Мемориска адреса на еден прозорец |
| DC | Device Context | - |
| DWM | Desktop Window Manager | - |

# Вовед

Во овој магистерски труд ќе биде прикажано креирање на свој сопствен UI framework користејќи познати библиотеки, графички API и познати техники, методи и пристапи за постигање на тие цели. UI framework претставува апстракција во која се обезбедува генеричка функционалност и може да се менува со дополнителен код. Еден UI framework обезбедува стандарден начин за градење и распоредување на GUI апликации и е софтверска средина за долгорочна употреба која обезбедува посебна функционалност како дел од поголема софтверска платформа за олеснување на развојот на софтверски апликации. Кога се користат библиотеки, често пати се користат само дел од нив, и со тоа, големината на самата крајна апликација може да се зголеми драстично и да предизвика проблем.

При креирањето на еден UI framework, неизбежно е да се наиде на поимот компјутерска графика. Компјутерска графика се занимава со генерирање слики и уметност. Компјутерската графика е основна технологија во повеќето специјализирани апликации. Развиен е голем дел од специјализиран хардвер и софтвер, при што приказите на повеќето уреди се управувани од компјутерски графички хардвер. Во овој труд, во темите од областа на компјутерската графика се користи за дизајн на кориснички интерфејс, спрајт графика, рендерирање, геометрија, компјутерска анимација, векторска графика, шејдери, обработка на слики, итн.

Windows обезбедува многу примитивни алатки за градење кориснички интерфејси. Системот обезбедува неколку основни контроли и природни контејнери за прозорци, но изградбата на сопствен кориснички интерфејс е тешка.

Корисничкиот интерфејс (UI) во презентираниот пример е воден според дизајн документите на софтверот Chromium и нивниот прилагоден views UI framework за прикажување на “chrome” UI делот на прелистувачот. Бидејќи Chromium тимот сакале различна естетика за Chromium прелистувачите, морале да изградат посебен framework на Windows за да го забрзаат развојот на приспособен, односно прилагоден интерфејс. Chromium користи Skia за сите графички операции.

Во дизајнот на UI framework, корисничкиот интерфејс првенствено се фокусира на информациската архитектура. Тоа е процес на градење интерфејси кои јасно му пренесуваат на корисникот што е важно во даден момент. Дизајнот на интерфејсот се

однесува на графички кориснички интерфејси и други форми на дизајн на интерфејс. Целта на дизајнот на корисничкиот интерфејс е да ја направи интеракцијата на корисникот што е можно поедноставна и поефикасна, во смисла на остварување на корисничките цели.

Графичкиот кориснички интерфејс (GUI), е форма на кориснички интерфејс што им овозможува на корисниците да комуницираат со електронски уреди преку графички икони и визуелни индикатори. Дејствата или настаните во GUI обично се изведуваат преку директна манипулација со графичките елементи.

Графичка библиотека е библиотека дизајнирана да помогне во прикажување на компјутерска графика на мониторот. Вклучува оптимизирани верзии на функции кои се справуваат со вообичаени задачи за рендерирање. Истото може да се изведе чисто во софтвер и да работи на процесорот, вообичаено во вградените системи или да биде хардверски забрзан од графичкиот процесор. Со користење на тие функции, програмата може да состави слика што треба да се емитува на мониторот.

Една од таквите библиотеки е наречена Skia и е користена во овој магистерски труд, при креирањето на еден UI framework. Skia е 2Д графичка библиотека со отворен код напишана во C++ која се фокусира на цртање, наместо да обезбедува свои графички контроли и јазик за опис на интерфејсот, итн.

Skia ги апстрахира графичките API специфични за платформата, олеснувајќи ја задачата да се научат природните графички API за платформата и нејзината специфична употреба. Skia исто така е користена од поголеми софтвери и frameworks и заради овие причини, беше главен кандидат за избор на графичка библиотека, за помош на креирање на еден UI framework. Исто така, поради овие причини, беше избран за полесно споредување на исцртаните резултати на екран со поголемите софтвери, како Chromium прелистувачот.

Во овој труд, во практичниот дел се прикажани во чекори подетално сите потребни фази за креирање на еден сопствен, прилагоден UI framework, кој може да служи и да се доразвива на специфичен начин согласно потребите на корисниците за креирање на апликации. Како што во денешен свет постојат “Chromium-based” прелистувачи, така, со употреба и следење на чекорите и фазите на практичниот дел, може да се дојде до свет каде што ќе произлезат прелистувачи, за различни потреби, коишто би се нарекувале “Skia-based” прелистувачи или апликации.

# 1. Софтверски framework (рамка)

Во компјутерското програмирање, софтверскиот framework (рамка) е апстракција во која софтверот, кој обезбедува генеричка функционалност, може селективно да се менува со дополнителен код напишан од корисникот, со што се обезбедува софтвер специфичен за апликацијата замислена од страна на корисникот. Еден софтверски framework (рамка) обезбедува стандарден начин за градење и распоредување на апликации и е универзална софтверска средина за повеќекратна употреба која обезбедува посебна функционалност како дел од поголема софтверска платформа за олеснување на развојот на софтверски апликации, производи и решенија.[1]

Софтверскиот framework (рамка) може да вклучува програми за поддршка, компајлери, библиотеки со кодови, збирки алатки и интерфејси за програмирање на апликации (API) кои ги обединуваат сите различни компоненти за да овозможат развој на проект или систем. [2]

Софтверските рамки носат карактеристики кои ги разликуваат од другите библиотеки и извори:

- Контрола на проток на извршување: во рамка, за разлика од библиотеките или во стандардните кориснички апликации, целокупниот проток на контрола на програмата не е диктиран од повикувачот, односно корисникот или клиентот, туку од рамката.
- Стандардно однесување: Може да се обезбеди со преоптоварување на методи во апстрактна класа која е обезбедена од рамката.
- Проширливост: Корисникот може да ја прошири рамката – обично со преоптоварување на класи обезбедени од самата рамка, при што програмерите можат да го додадат свој кориснички код за да обезбедат специфична функционалност. Ова обично се постигнува со виртуелен метод на класа што го преоптоварува обичниот метод на подкласата.

[1] https://en.wikipedia.org/wiki/Software_framework
[2] https://en.wikipedia.org/wiki/Software_framework

- Непроменлив код од рамката: кодот од самата рамка, генерално, не треба да се менува или не може, додека на некои делови од протокот на извршување прифаќа екстензии имплементирани од корисникот. Со други зборови, корисниците можат да ја прошират рамката, но не можат да го менуваат нејзиниот код или нејзиното генерално однесување.

Целите на дизајнерите на еден софтверски framework имаат задача да го олеснат развојот на софтверот дозволувајќи им на дизајнерите и програмерите на апликацијата кои го користат истиот framework да го посветат своето време на исполнување на барањата за нивниот софтвер наместо да се занимаваат со постандардните детали на ниско ниво за обезбедување на работен систем и усогласување со постојните стандарди, со што се намалува целокупното време за развој.[3]

Користењето на framework често ја зголемува големината на програмите, феномен што се нарекува „code bloat“. Поради потребите на апликациите барани од клиентите, и конкурентните и комплементарните frameworks знаат да бидат искористени во еден продукт. Поради сложеноста на нивните API, планираното намалување на целокупното време за развој може да не се постигне поради потребата да се троши дополнително време за учење за правилно користење на рамката; оваа критика е валидна кога развојниот персонал првпат ќе наиде на посебна или нов framework. Доколку таквиот framework не се користи во следните работни задачи, времето вложено во учењето на рамката може да чини повеќе од постојниот напишан код, познат на персоналот на проектот; многу програмери чуваат копии од корисен код за заеднички потреби.[4]

Меѓутоа, штом ќе се научи framework (рамката), идните проекти може да бидат побрзи и полесни за завршување; концептот на еден framework е да се направи сет на решенија што одговараат на сите, а со запознавање, кодот логично ќе се зголеми. Релативната ефикасност и концизност на кодот не зависи од неговата големина. Користењето на библиотеки често доведува до користење само дел од нив, и со тоа,

[3] https://en.wikipedia.org/wiki/Software_framework
[4] https://en.wikipedia.org/wiki/Software_framework

големината на самата крајна апликација да се зголеми драстично и да предизвика проблем.[5]

Најефективните frameworks се оние што еволуираат од рефакторирање на заеднички, отворен код, наместо да се користи генерички framework „наменет за сите“ развиен од трети страни за општи цели. Корисничкиот интерфејс во таков пакет на апликации има заеднички изглед, корисничко искуство, атрибути и методи за споделување на податоци.[6]

# 2. Апликациски framework (рамка)

Во компјутерското програмирање, рамката за апликација се состои од софтверски framework што го користат развивачите на софтвер за да ја имплементираат стандардната структура на апликативниот софтвер.

**Историја**

Апликациските frameworks станаа популарни со подемот на графичките кориснички интерфејси (GUI), бидејќи тие имаат тенденција да промовираат стандардна структура за апликации. Програмерите сметаат дека е многу поедноставно да се создадат алатки за автоматско креирање GUI кога користат стандардна рамка, бидејќи ова однапред ја дефинира основната структура на кодот на апликацијата. Програмерите обично користат техники од објектно-ориентирано програмирање (OOP) за имплементација на нивниот софтвер, така што уникатните делови на апликацијата едноставно можат да наследат од класите што постојат во framework (рамката).[7]

Apple Computer меѓу првите развија framework за комерцијални апликации, MacApp (1985 година), за Macintosh. Првично напишана во проширена (објектно-ориентирана) верзија на Pascal наречена Object Pascal, подоцна беше препишана во C++. Друга значајна рамка за Mac е PowerPlant на Metrowerks, базирана на Carbon (API). Cocoa (framework) за macOS нуди поинаков пристап за апликации, базиран на framework (рамката) OpenStep развиена во NeXT.

[5] https://en.wikipedia.org/wiki/Software_framework
[6] https://en.wikipedia.org/wiki/Software_framework
[7] https://en.wikipedia.org/wiki/Application_framework

Од 2010 па наваму, многу апликации се создадени со framework базиран на проектот Chromium на Google. Исто така, истакнати се неколку frameworks, како што се Electron и Chromium Embedded Framework (CEF).

Бесплатните софтверски frameworks со отворен код постојат како дел од проектите на Mozilla, LibreOffice, GNOME, KDE, NetBeans и Eclipse.

Microsoft нуди framework за развој на Windows апликации во C++ наречена Microsoft Foundation Class Library (MFC), и сличен framework за развој на апликации со Visual Basic или C#, наречена .NET Framework.

Неколку frameworks можат да градат меѓуплатформски апликации за Linux, Macintosh и Windows од заеднички изворен код, како што се Qt, wxWidgets, Juce, Fox toolkit или Eclipse Rich Client Platform (RCP).

# 3. Компјутерска графика

Компјутерската графика се занимава со генерирање слики и уметност со помош на компјутери. Компјутерската графика е основна технологија во дигиталната фотографија, филмот, видео игрите, дигиталната уметност, мобилните телефони и компјутерите и многу специјализирани апликации. Развиен е голем дел од специјализиран хардвер и софтвер, при што приказите на повеќето уреди се управувани од компјутерски графички хардвер. Тоа е огромна и неодамна развиена област на компјутерската наука. Фразата е измислена во 1960 година од истражувачите на компјутерска графика Верн Хадсон и Вилијам Фетер од Боинг. Често се скратува како CG, или обично во контекст на филмот како компјутерски генерирани слики (CGI). Неуметничките аспекти на компјутерската графика се предмет на истражување на компјутерската наука.[8]

Некои теми во компјутерската графика вклучуваат дизајн на кориснички интерфејс, спрајт графика, рендерирање, ray tracing, обработка на геометрија, компјутерска анимација, векторска графика, 3Д моделирање, шејдери (нијансирање и ефекти, програми што се извршуваат од страна на графичкиот хардвер), графички процесорски дизајн, имплицитни површини, визуелизација, научно пресметување, обработка на слики, пресметковна фотографија, научна визуелизација, геометрија, компјутерска визија и друго. Целокупната

[8] „ACM Computing Classification System ToC“. Здружение за компјутерски машини. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Computer_graphics

методологија во голема мера зависи од основите за геометријата, оптиката, физиката и перцепцијата.

Компјутерската графика е одговорна за ефикасно и значајно прикажување на податоците за уметност и слика на потрошувачот. Се користи и за обработка на податоци за слики добиени од физичкиот свет, како што се фотографии и видео содржини. Развојот на компјутерска графика имаше значително влијание врз многу видови медиуми и доведе до револуција во анимацијата, филмовите, рекламирањето и видео игрите[9].

Терминот компјутерска графика се однесува на неколку работи:

- претставување и манипулација со податоците од сликата од страна на компјутер;
- различните технологии кои се користат за креирање и манипулирање со слики;
- методи за дигитално синтетизирање и манипулирање со визуелна содржина.

Развиени се многу алатки за визуелизација на податоците. Сликите генерирани од компјутер може да се категоризираат во неколку различни типови: дводимензионална (2Д), тридимензионална (3Д) и анимирана графика. Како што се подобрува технологијата, 3Д компјутерската графика стана почеста, но 2Д компјутерската графика сè уште е широко користена. Компјутерската графика се појави како посебна гранка на компјутерската наука која ги проучува методите за дигитално синтетизирање и манипулирање со визуелна содржина.

## 3.1 Видови на слики

**Дводимензионални**

2Д компјутерската графика е компјутерска генерација на дигитални слики - главно од модели, како што е дигитална слика и според техники специфични за нив.

2Д компјутерската графика главно се користи во апликации кои првично беа развиени врз традиционални технологии за печатење и цртање, како што е типографијата. Во тие апликации, дводимензионалната слика не е само претстава на објект од реалниот свет, туку независен артефакт со додадена семантичка вредност; Затоа се претпочитаат дводимензионални модели бидејќи даваат подиректна контрола на сликата отколку 3Д

[9] https://en.wikipedia.org/wiki/Computer_graphics

компјутерската графика, чиј пристап е повеќе сличен на фотографија отколку на типографија.[10]

**Пикселна уметност**

Голема форма на дигитална уметност, пикселната уметност се создава преку употреба на растерски графички софтвер, каде сликите се уредуваат на ниво на пиксели. Графиките во повеќето стари (или релативно ограничени) компјутерски и видео игри, игрите со калкулатори со графикони и многу игри за мобилни телефони се главно пикселна уметност. [11]

**Спрајт графика**

Sprite е дводимензионална слика или анимација која е интегрирана во поголема сцена. Првично вклучуваше само графички објекти со кои се ракува одделно од мемориската битмапа на видео приказ, ова вклучува различни начини на графички преклопувања.

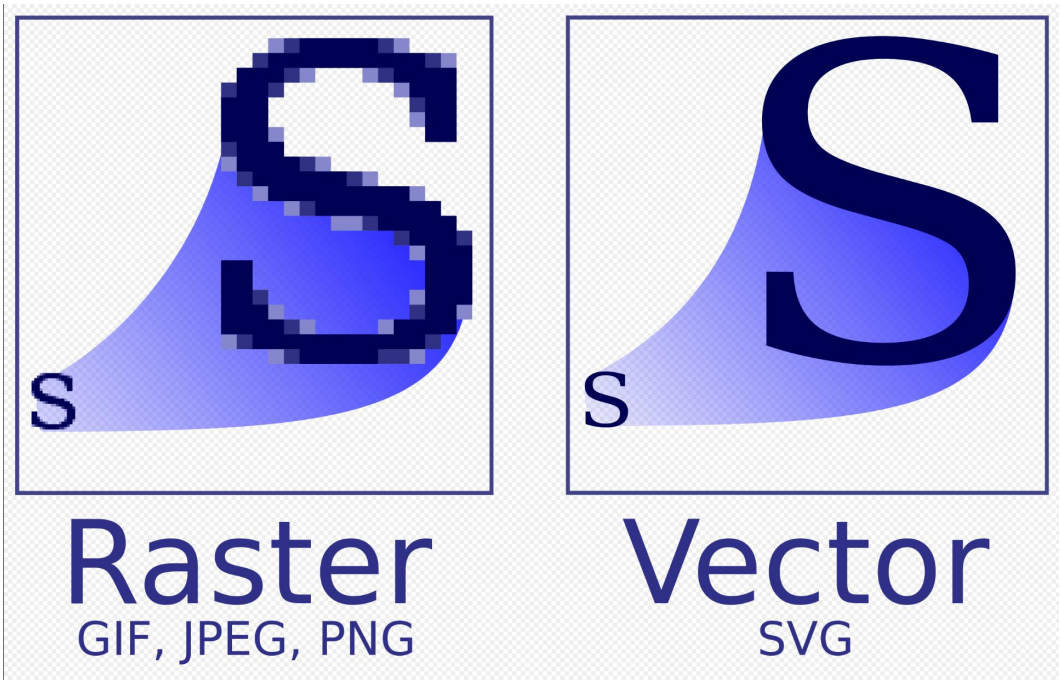
Првично, sprites беа метод за интегрирање на неповрзани битмапи, така што тие изгледаа како дел од нормалната битмапа на екранот, како што е создавање на анимиран лик што може да се премести на екранот без да се менуваат податоците што го дефинираат целокупниот екран. [12]

**Векторска графика**

Форматите на векторската графика се комплементарни со растерската графика. Растерската графика е претставување на сликите како низа од пиксели и обично се користи за прикажување на фотографски слики (види слика 1).[13] Векторската графика се состои од кодирање на информации за облиците и боите што ја сочинуваат сликата, што може да овозможи поголема флексибилност во прикажувањето. Има случаи кога работата со векторски алатки и формати е најдобра практика, а други случаи кога е обратно, односно наместо векторски, да се користи растерски. Има моменти кога двата формати се

[10] https://en.wikipedia.org/wiki/Computer_graphics
[11] https://en.wikipedia.org/wiki/Computer_graphics
[12] https://en.wikipedia.org/wiki/Computer_graphics
[13] Greenberg, Ira (2007). Processing: Creative Coding and Computational Art. Apress. ISBN 978-1-59059-617-3. - https://en.wikipedia.org/wiki/Computer_graphics

спојуваат. Разбирањето на предностите и ограничувањата на секоја технологија и односот меѓу нив најверојатно ќе резултира со ефикасна и ефективна употреба на алатките.




*Слика 1. Растерска и наспроти векторска слика кога ќе се претстави на поголема резолуција*

**Генеративни модели за машинско учење**

Од средината на 2010, како резултат на напредокот во длабоките невронски мрежи, беа создадени модели кои како влез земаат опис на природен јазик и како излез произведуваат слика што одговара на тој опис. Моделите текст-во-слика генерално комбинираат јазичен модел, кој го трансформира влезниот текст во латентна репрезентација и генеративен модел на слика, кој произведува слика условена од таа репрезентација. Најефективните модели обично се обучени за огромни количини на податоци за слики и текст избришани од мрежата. До 2022 година, најдобрите од овие модели, како Dall-E 2, Stable Diffusion и Midjourney, ќе можат да создаваат слики во низа стилови, кои се движат од имитација на живи уметници до речиси фотореалистички, за неколку секунди, со оглед на доволно моќни хардвери.[14]

[14] Винсент, Џејмс (2022). „All these images were generated by Google's latest textto-image AI“. The Verge. Vox Media. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Computer_graphics

**Тридимензионални**

3Д графика, во споредба со 2Д графика, е графика која користи тродимензионално претставување на геометриски податоци. Заради перформанси, геометриските податоци се складирани на компјутерот. Ова вклучува слики што може да бидат подоцна прикажани или гледање во реално време.

И покрај овие разлики, 3Д компјутерската графика се потпира на слични алгоритми кои се користат во 2Д компјутерската графика и растерската графика за да постигне конечен рендериран приказ. Во софтверот за компјутерска графика, разликата помеѓу 2Д и 3Д повремено е нејасна; 2Д апликациите може да користат 3Д техники за да постигнат ефекти како осветлување, а првенствено 3Д може да користат техники за 2Д рендерирање.

3Д компјутерската графика е иста како и 3Д моделите. Моделот е содржан во датотеката со графички податоци, освен во рендерирањето. Сепак, постојат разлики кои вклучуваат 3Д модел да биде „сцена" или самата репрезентација на кој било 3Д објект.[15]

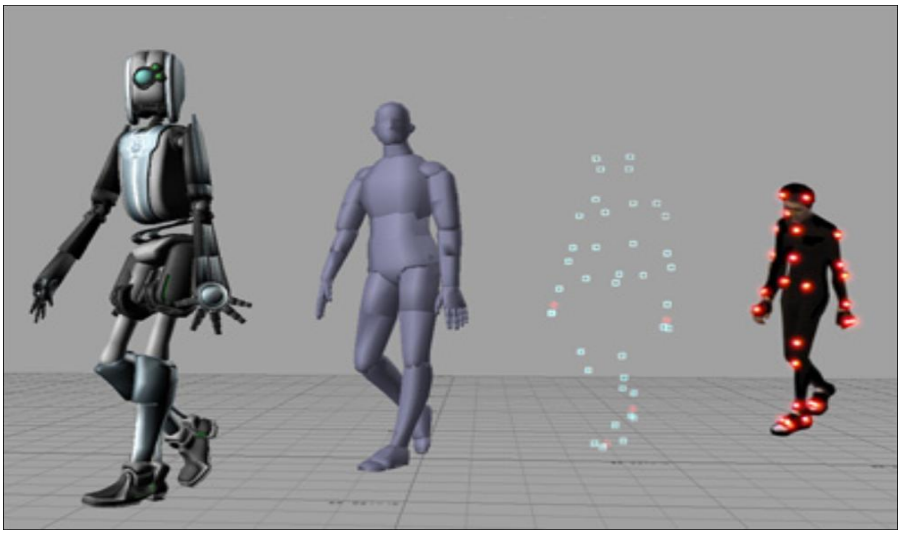
**Компјутерска анимација**

Компјутерската анимација е уметност на создавање подвижни слики преку употреба на компјутери. Тоа е подобласт на компјутерска графика и анимација. Сè повеќе се создава со помош на 3Д компјутерска графика, иако 2Д компјутерската графика сè уште е широко користена за стилски, низок пропусен опсег и побрзи потреби за рендерирање во реално време. Понекогаш целта на анимацијата е друг медиум, како што е филмот, и тогаш компјутерската анимација се нарекува CGI (компјутерски генерирани слики).

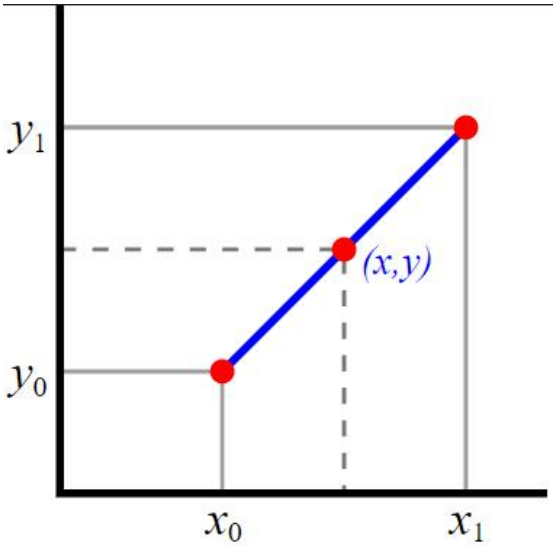
Виртуелните ентитети може да содржат и да бидат контролирани од различни атрибути, како што се вредностите на трансформацијата (локација, ротација и големина) складирани во матрицата за трансформација на објектот. **Анимација е промена на атрибут со текот на времето**. Постојат повеќе методи за постигнување анимација; рудиментарната форма се заснова на создавање и уредување на "keyframes", при што секоја складира вредност во дадено време, по атрибут што треба да се анимира. Софтверот за 2Д/3Д графика ќе се менува со секоја клучна рамка, создавајќи крива на вредност што може да се уредува со текот на времето, што резултира со анимација.[16]

[15] https://en.wikipedia.org/wiki/Computer_graphics
[16] https://en.wikipedia.org/wiki/Computer_graphics

За да се создаде илузија на движење, на компјутерскиот екран се прикажува слика, а потоа брзо се заменува со нова слика која е слична на претходната слика, но малку поместена. Оваа техника е идентична со илузијата на движење во телевизијата и филмовите.



*Слика 2. Пример за компјутерска анимација произведена со "Motion Capture"*




*Слика 3. Линеарна интерполација*[17]

Формула за линеарна интерполација помеѓу 2 точки:

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0},$$

[17] https://en.wikipedia.org/wiki/Linear_interpolation

За дадена конечна вредност x (t), има пораст на y, и според дадената вредност од x, се добива следната вредност на y според x. Користејќи ја формулата за линеарна интерполација, анимацијата се движи по која било крива за да x ја пресмета вредноста на y.

## 3.2 Концепти и принципи

Сликите обично се создаваат од уреди како што се камери, огледала, леќи, телескопи, микроскопи, итн.

Дигиталните слики вклучуваат и векторски и растерски слики, но растерските слики почесто се користат.

**Пиксел**

Во дигиталната слика, пиксел е една точка во растерска слика. Пикселите се поставени на обична 2-димензионална мрежа и често се претставени со помош на точки или квадрати. Секој пиксел е примерок од оригинална слика, каде што повеќе примероци обично обезбедуваат попрецизна претстава на оригиналот. Интензитетот на секој пиксел е променлив; во системите во боја, секој пиксел обично има три канали како црвена, зелена и сина.

Графиките се визуелни претстави на површина, како што е компјутерскиот екран. Примери се фотографии, цртежи, графички дизајни, мапи, инженерски цртежи или други слики. Графиката често комбинира текст и илустрација. Графичкиот дизајн може да се состои од избирање, креирање или распоредување на типографија, како во брошура, флаер, постер, веб-страница или книга без релевантен елемент.[18]

**Основни елементи (примитивни)**

Примитивците се основни единици кои графичкиот систем може да ги комбинира за да создаде посложени слики или модели. Примери би биле sprites во 2Д видео игри, геометриски примитиви во CAD или полигони или триаголници во 3Д прикажување. Примитивците може да бидат поддржани во хардвер за ефикасно прикажување, или градежни блокови обезбедени од графичка апликација или framework.[19]

[18] https://en.wikipedia.org/wiki/Computer_graphics
[19] https://en.wikipedia.org/wiki/Computer_graphics

**Рендерирање**

Рендерирање е генерирање на 2Д слика од 3Д модел со помош на компјутерски програми. Датотеката со сцена содржи објекти во строго дефиниран јазик или структура на податоци; кои содржат геометрија, гледна точка, текстурирање, осветлување и информации за засенчување како опис на виртуелната сцена.[20] Податоците содржани во датотеката со сцената потоа се пренесуваат до програма за рендерирање што треба да се обработи и да се емитува во дигитална слика или датотека со растерска графичка слика. Програмата за рендерирање обично е вградена во софтверот за компјутерска графика, иако други се достапни како приклучоци или целосно посебни програми. Иако техничките детали за методите на рендерирање варираат, општите предизвици што треба да се надминат при производството на 2Д слика од 3Д претстава зачувана во датотека со сцена се пренесени преку графичкиот pipeline до уредот за рендерирање, како што е графичкиот процесор. GPU е уред кој може да му помогне на процесорот во пресметките. Ако сцената треба да изгледа релативно реална и предвидлива под виртуелно осветлување, софтверот за рендерирање треба да ја реши равенката за рендерирање. Равенката за рендерирање не ги зема предвид сите појави на осветлување, туку е општ модел на осветлување за компјутерски генерирани слики. „Рендерирање“ се користи и за да се опише процесот на пресметување на ефектите во датотека за уредување видео за да се произведе финална слика.

## 3.3 Шејдер (shader)

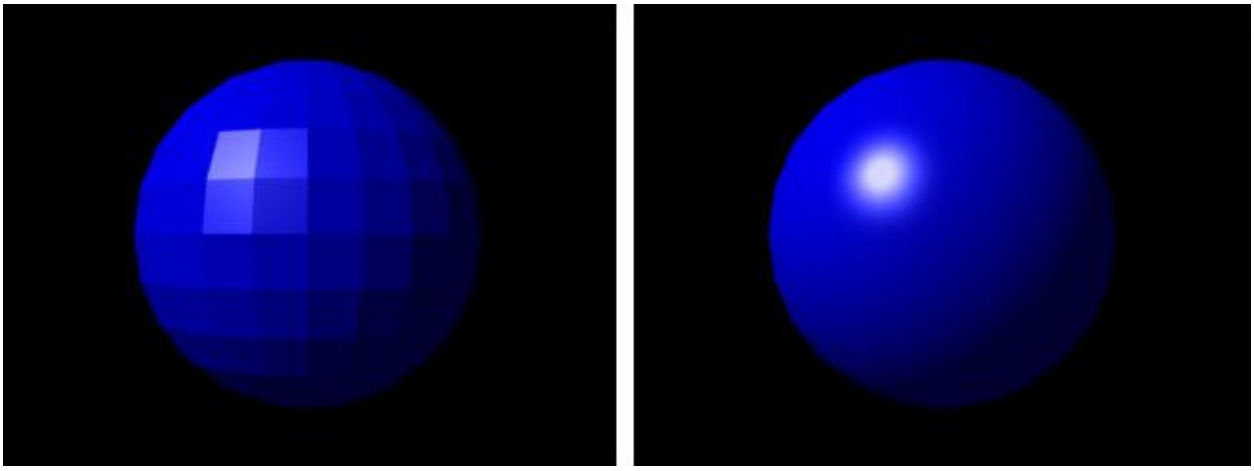
Во компјутерската графика, шејдер е компјутерска програма која ги пресметува соодветните нивоа на светлина, темнина и боја за време на прикажувањето на 3Д сцена - процес познат како засенчување. Шејдерите еволуираа за да вршат различни специјализирани функции во компјутерска графика со специјални ефекти и пост-процесирање на видео, како и општонаменско пресметување на графички процесорски единици.

Традиционалните шејдери пресметуваат ефекти на рендерирање на графичкиот хардвер со висок степен на флексибилност. Повеќето шејдери се кодирани за (и работат на)

[20] „Lighting principles for 3D artists from film and art“. GarageFarm. Преземено на 2024-06-06.- https://en.wikipedia.org/wiki/Computer_graphics

графичка процесорска единица (GPU)[21], но теоретски истите можат да се извршат од страна на CPU. Јазиците за засенчување се користат за програмирање на цевководот, односно pipeline за рендерирање на графичкиот процесор, кој главно го заменува цевководот со фиксна функција од минатото кој дозволувал само вообичаени функции за трансформација на геометрија и засенчување на пиксели. Со шејдери, се овозможува да се користат приспособени ефекти. Позицијата и бојата (нијанса, заситеност, осветленост и контраст) на сите пиксели, темиња и/или текстури што се користат за конструирање на конечна рендерирана слика може да се менуваат со користење на алгоритми дефинирани во шејдер и може да се менуваат со надворешни променливи или текстури воведена од компјутерската програма која го повикува шејдерот.

Покрај едноставните модели на осветлување, посложените употреби на шејдерите вклучуваат: промена на нијансата, заситеноста, осветленоста или контрастот на сликата; производство на заматување, светло, волуметриско осветлување, нормално мапирање (за ефекти на длабочина), засенчување на ќелиите, постеризација, изобличување, откривање на рабови и движење, итн.



*Слика 4. Пример за два вида шејдери: рамно засенчување лево и Phong засенчување надесно*[22]

[21] „LearnOpenGL - Shaders“. Learnopengl.com. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Shader
[22] https://en.wikipedia.org/wiki/Shader



*Слика 5. Друга употреба на шејдерите е за специјални ефекти, дури и на 2Д слики, (на пр., фотографија од веб-камера). Непроменетата, незасенчена слика е лево, а истата слика има шејдер применет десно*[23]

Шејдерите се едноставни програми кои ги опишуваат карактеристиките на темето или на пикселот. Vertex Shaders ги опишуваат атрибутите (позиција, координати на текстура, бои, итн.) на темето, додека Pixel Shaders ги опишуваат карактеристиките (боја, z-длабочина и алфа вредност) на пикселот. Се повикува Vertex Shader за секое теме во примитив (можно по теселација); на начин: едно теме како *input*, едно (ажурирано) теме како *output*. Секое теме потоа се прикажува како серија од пиксели на површина (блок меморија) која на крајот ќе биде испратена на екранот.[24]

Основниот графички цевковод е како што следува:

- Процесорот испраќа инструкции (компајлирани програми за јазик за засенчување) и податоци за геометријата до графичката картичка.
- Во рамките на Vertex Shader, геометријата се трансформира.
- Ако шејдерот на геометрија е во единицата за графичка обработка и е активен, се вршат одредени промени на геометриите во сцената.
- Ако Tessellation Shader е во единицата за графичка обработка и е активен, геометриите во сцената може да се поделат.
- Пресметаната геометрија се дели на триаголници.

[23] https://en.wikipedia.org/wiki/Shader
[24] https://en.wikipedia.org/wiki/Shader

- Триаголниците се поделени на фрагменти во четворки (еден фрагмент четириаголник е примитивен фрагмент од 2 × 2).
- Четворките на фрагментите се модифицирани според Fragment Shader.
- Тестот за длабочина се врши; фрагментите што поминуваат ќе бидат запишани на екранот и може да се помешаат во баферот на рамката.

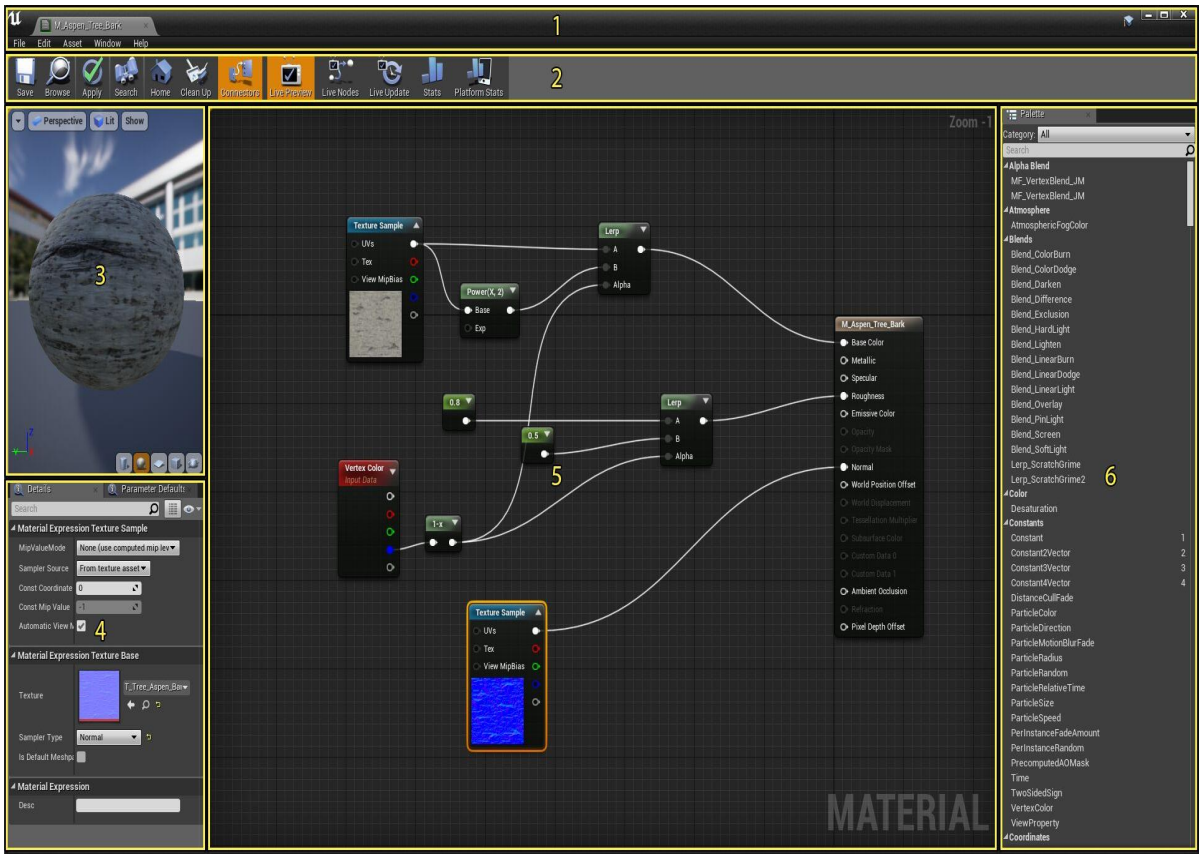
Графичкиот цевковод, односно pipeline ги користи овие чекори со цел да ги трансформира тридимензионалните (или дводимензионалните) податоци во корисни дводимензионални податоци за прикажување. Во принцип, ова е голема матрица на пиксели или „бафер на рамка“, односно “frame buffer”.[25]

Јазикот на кој се програмираат шејдерите зависи од околината. Официјалниот јазик за засенчување OpenGL и OpenGL ES е OpenGL Shading Language, познат и како GLSL, а официјалниот јазик за засенчување Direct3D е High Level Shader Language, познат и како HLSL. Apple објави свој јазик за засенчување наречен Metal Shading Language како дел од Metal.[26]

**Уредувачи на шејдери преку GUI**

Современите платформи за развој на видео игри како Unity, Unreal Engine и Godot сè повеќе вклучуваат уредници базирани на јазли кои можат да креираат шејдери без потреба од вистински код. Наместо тоа, на корисникот му се прикажува насочен график на поврзани јазли што им овозможува на корисниците да насочуваат различни текстури, мапи и математички функции во излезни вредности како што се дифузната боја, спекуларната боја и интензитет, грубост/металност, висина, нормална итн. Автоматската компилација потоа го претвора графикот во вистински, компајлиран шејдер.[27]

[25] https://en.wikipedia.org/wiki/Shader
[26] https://en.wikipedia.org/wiki/Shader
[27] https://en.wikipedia.org/wiki/Shader




*Слика 6. Уредувачот на шејдери во Unreal Engine 4. Интерфејсот на уредувачот на шејдери се состои од лента со мени, лента со алатки и четири отворени панели*[28]

# 4. Дизајн на кориснички интерфејс

Дизајнот на корисничкиот интерфејс (UI) или инженерството на корисничкиот интерфејс е дизајн на кориснички интерфејси за машини и софтвер, како што се компјутери, домашни апарати, мобилни уреди и други електронски уреди, со фокус на максимизирање на употребливоста и корисничкото искуство. Во дизајнот на компјутер или софтвер, дизајнот на корисничкиот интерфејс (UI) првенствено се фокусира на информацискатa архитектура. Тоа е процес на градење интерфејси кои јасно му пренесуваат на корисникот што е важно. Дизајнот на интерфејсот се однесува на графички кориснички интерфејси и други форми на дизајн на интерфејс. Целта на дизајнот на корисничкиот интерфејс е да ја направи интеракцијата на корисникот што е можно

[28] https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/Materials/Editor/Interface/

поедноставна и поефикасна, во смисла на остварување на корисничките цели (дизајн фокусиран на корисникот).[29]

Корисничките интерфејси се точките на интеракција помеѓу корисниците и дизајните. Постојат три вида:

1. **Графички кориснички интерфејси (GUI)** – Корисниците комуницираат со визуелни претстави на екранот на компјутерот. Десктопот е пример за GUI.
2. Интерфејси контролирани преку глас.
3. Интерактивни интерфејси кои користат гестови.

Дизајнерите на кориснички интерфејси имаат тенденција да се специјализираат за одредени видови проекти и имаат вештини фокусирани на нивната експертиза, без разлика дали се работи за дизајн на софтвер, истражување на корисници, веб-дизајн или индустриски дизајн.

**Во споредба со UX дизајн**

Во споредба со UX дизајн, дизајн на UI е повеќе за површината и целокупниот изглед на дизајнот. Дизајнот на корисничкиот интерфејс е занает во кој дизајнерите вршат важна функција во креирањето на корисничкото искуство. Дизајнот на интерфејсот треба да ги информира корисниците за тоа што се случува, давајќи соодветни повратни информации, во секое дадено време. Визуелниот изглед и чувство на дизајнот на интерфејсот го поставува тонот за корисничкото искуство.[30] Од друга страна, терминот UX дизајн се однесува на целиот процес на создавање корисничко искуство.

# 5. Графички кориснички интерфејс (GUI)

Графички кориснички интерфејс, или GUI, е форма на кориснички интерфејс што им овозможува на корисниците да комуницираат со електронски уреди преку графички икони и визуелни индикатори. Во многу апликации, графичките интерфејси се користат наместо интерфејсите засновани на текст, кои се засноваат на означени ознаки со команди или навигација со текст. GUI беа воведени како реакција на воочената стрмна крива на

[29] https://en.wikipedia.org/wiki/User_interface_design
[30] Рот, Роберт Е. (2017). „User Interface and User Experience (UI/UX) Design" Географски информатички науки и технологија Тело на знаење. 2017 година (Q2). doi:10.22224/gistbok/2017.2.5. - https://en.wikipedia.org/wiki/User_interface_design

учење на интерфејсите на командната линија (CLIs)[31][32][33], кои бараат командите да се напишат на тастатурата на компјутерот.

Дејствата или настани во GUI обично се изведуваат преку директна манипулација со графичките елементи.[34][35][36] Покрај компјутерите, GUI се користат во многу рачни мобилни уреди како што се MP3 плеери, преносни медиа плеери, уреди за игри, паметни телефони и помали контроли за домаќинство, канцеларии и индустрии.

## 5.1 GUI и дизајн на интеракција

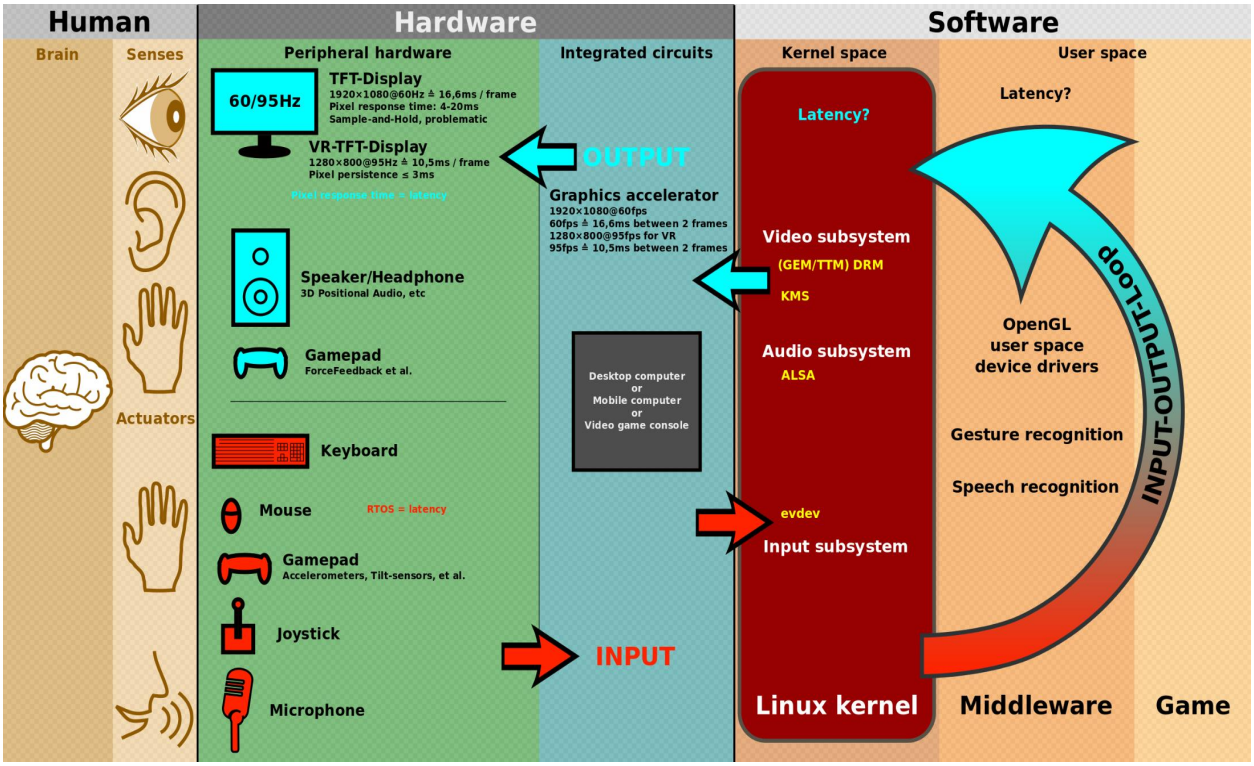
Дизајнирањето на визуелниот состав и временското однесување на GUI е важен дел од програмирањето на софтверска апликација во областа на интеракцијата човек-компјутер. Неговата цел е да ја подобри ефикасноста и леснотијата на користење за основниот логички дизајн на складирана програма, дизајнерска дисциплина наречена употребливост. Се користат методи на дизајн насочен кон корисникот за да се осигура дека визуелниот јазик воведен во дизајнот е добро прилагоден на задачите.

Видливите карактеристики на графичкиот интерфејс на апликацијата понекогаш се нарекуваат “chrome” или GUI.[37][38][39] Вообичаено, корисниците комуницираат со информации преку манипулирање на визуелни графички контроли кои овозможуваат интеракции соодветни на видот на податоци за таа контрола. Контролите на добро дизајниран интерфејс се дизајнирани за да ги поддржат дејствата неопходни за постигнување на целите на корисниците. Модел-приказ-контролер, односно **Model-View-**

---

[31] „Command line vs. GUI“. Computer Hope. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Graphical_user_interface
[32] "The GUI versus the Command Line: Which is better? (Part 1)". Операции на Microsoft.com. Microsoft Learn. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Graphical_user_interface
[33] "The GUI versus the Command Line: Which is better? (Part 2)". Операции на Microsoft.com. Microsoft Learn. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Graphical_user_interface
[34] „Graphical user interface“. ScienceDaily. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Graphical_user_interface
[35] Levy, Steven. "Graphical User Interface (GUI)". Britannica.com. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Graphical_user_interface
[36] „GUI“. PC Magazine Енциклопедија. pcmag.com. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Graphical_user_interface
[37] „chrome“. www.catb.org. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Graphical_user_interface
[38] Нилсен, Јакоб (2012). „Browser and GUI Chrome“. Nngroup. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Graphical_user_interface
[39] Мартинез, Венди Л. (2011). „Graphical user interfaces: Graphical user interfaces“. Wiley Interdisciplinary Reviews: Computational Statistics. 3 (2): 119–133. doi:10.1002/wics.150. S2CID 60467930. - https://en.wikipedia.org/wiki/Graphical_user_interface

**Controller (MVC)** дозволува флексибилни структури во кои интерфејсот е независен од и индиректно поврзан со функциите на апликацијата, така што GUI може лесно да се приспособи. Ова им овозможува на корисниците да изберат или дизајнираат различен стил или тема по своја волја и ја олеснува работата на дизајнерот да го промени интерфејсот како што еволуираат потребите на корисниците. Добриот GUI дизајн повеќе се однесува на корисниците, а помалку на системската архитектура. Големите контроли, како што се прозорците, обично обезбедуваат некоја рамка или контејнер за главната содржина на презентацијата, како што се веб-страница, е-порака или цртеж.




*Слика 7. GUI е претставен (прикажан) на екранот на компјутерот. Резултат на обработен влез на корисникот и главниот интерфејс за интеракција човек-машина*[40]

**Компоненти**

GUI користи комбинација од технологии и уреди за да обезбеди платформа со која корисниците можат да комуницираат, за задачите на собирање и производство на информации.

Низа елементи што одговараат на визуелен јазик еволуирале за да претставуваат информации зачувани во компјутерите. Ова им олеснува на луѓето со малку компјутерски

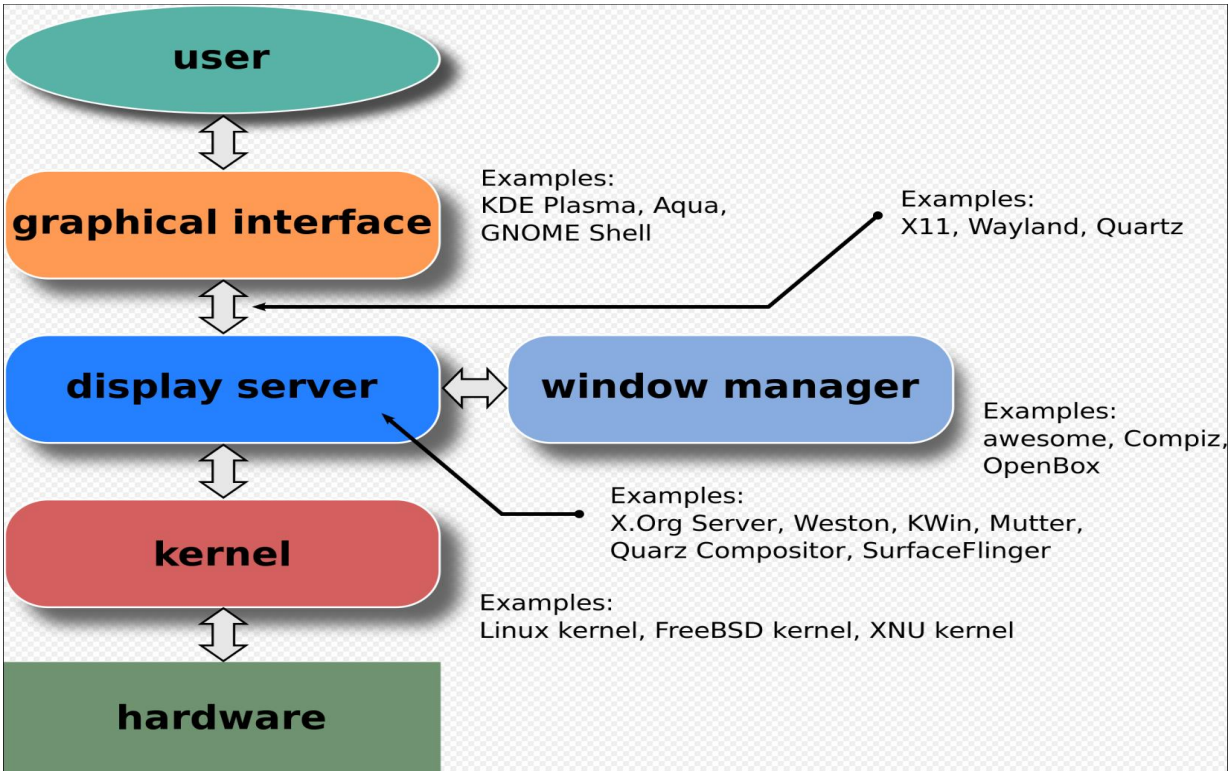
[40] https://en.wikipedia.org/wiki/Graphical_user_interface

вештини да работат и да користат компјутерски софтвер. Најчестата комбинација на такви елементи во GUI се прозорците, иконите, текстуалните полиња, платната, менијата, парадигмата на покажувачот (WIMP), особено кај персоналните компјутери.[41]

Стилот на интеракција WIMP користи виртуелен влезен уред за да ја претстави позицијата на интерфејсот на уредот за покажување, најчесто глушец, и прикажува информации организирани во прозорци и претставени со икони. Достапните команди се компајлираат заедно во менијата, а дејствата се изведуваат правејќи гестови со уредот за покажување. Управувачот со прозорци ги олеснува интеракциите помеѓу прозорците, апликациите и системот за прозорци. Системот за прозорци се справува со хардверски уреди како што се уреди за покажување, графички хардвер и позиционирање на покажувачот.

Во персоналните компјутери, сите овие елементи се моделираат преку метафора за да се произведе симулација наречена десктоп околина во која екранот претставува работна површина, на која може да се постават документи и папки со документи. Управувачите со прозорци и другиот софтвер се комбинираат за да ја симулираат работната околина со различен степен на реализам.

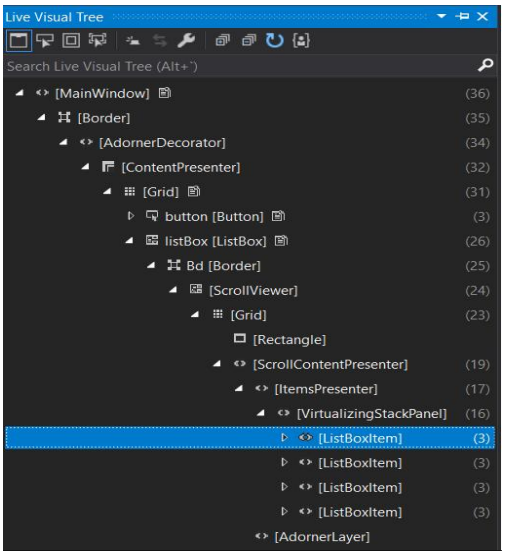
[41] "What is a graphical user interface (GUI)?". IONOS Digitalguide. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Graphical_user_interface




*Слика 8. Слоеви на GUI базирани на систем за прозорци*[42]

## 5.2 GUI дрво

GUI дрво е график кој ги визуелизира сите GUI компоненти на софтверот. Секој јазол од дрвото означува една од GUI компонентите што се користат од една апликација.[43]




*Слика 9. GUI дрво*

[42] https://en.wikipedia.org/wiki/Graphical_user_interface
[43] https://en.wikipedia.org/wiki/Integration_tree

# 6. Графичка библиотека или графичко API

Графичка библиотека е програмска библиотека дизајнирана да помогне во прикажувањето на компјутерската графика на мониторот. Обично вклучува обезбедување оптимизирани верзии на функции кои се справуваат со вообичаени задачи за рендерирање. Истото може да се изведе чисто во софтвер и да работи на процесорот, вообичаено во вградените системи или да биде хардверски забрзан од графичкиот процесор, што е почеста кај компјутерите. Со користење на тие функции, програмата може да состави слика што треба да се емитува на мониторот. Тоа го ослободува програмерот од задачата да ги креира и оптимизира тие функции и му овозможува да се фокусира на градење на графичката програма.[44]

3Д графиката стана толку популарна, што се создадени специјализирани API за да ги олеснат процесите во сите фази на генерирање компјутерска графика. Овие API се покажаа како витални за производителите на хардвер за компјутерска графика, бидејќи обезбедуваат начин за да програмерите  пристапат до хардверот на апстрактен начин, додека сè уште го користат специјалниот хардвер на која било специфична графичка картичка.[45]

**OpenGL**

OpenGL (Open Graphics Library[46]) е меѓујазичен, меѓуплатформски апликативен програмски интерфејс (API) за прикажување на 2Д и 3Д векторска графика. OpenGL е управуван од непрофитниот технолошки конзорциум Khronos Group[47].

OpenGL опишува апстрактен интерфејс за програмирање на апликации за цртање 2Д и 3Д графика. Дизајниран е да се имплементира главно или целосно со користење на хардверско забрзување како што е графичкиот процесор, иако е можно да се имплементира целосно во софтвер што работи на процесорот.

API е дефиниран како збир на функции кои може да се повикаат од клиентската програма, заедно со множество именувани целобројни константи. Иако дефинициите на

---

[44] https://en.wikipedia.org/wiki/Graphics_library
[45] https://en.wikipedia.org/wiki/List_of_3D_graphics_libraries
[46] „The OpenGL Graphics System: A Specification“ (PDF). 4.0 (Основен профил). Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/OpenGL
[47] „OpenGL ARB to Pass Control of OpenGL Specification to Khronos Group“. Групата Хронос. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/OpenGL

функциите се површно слични со оние на програмскиот јазик C, тие се независни од јазикот.

Покрај тоа што е независен од јазикот, OpenGL е и меѓуплатформски. Спецификацијата не кажува ништо за темата за добивање и управување со контекст на OpenGL, оставајќи ја таа обврска на повикувачот. Од истата причина, OpenGL чисто се занимава со рендерирање, не обезбедувајќи API поврзани со влез, аудио или прозор.

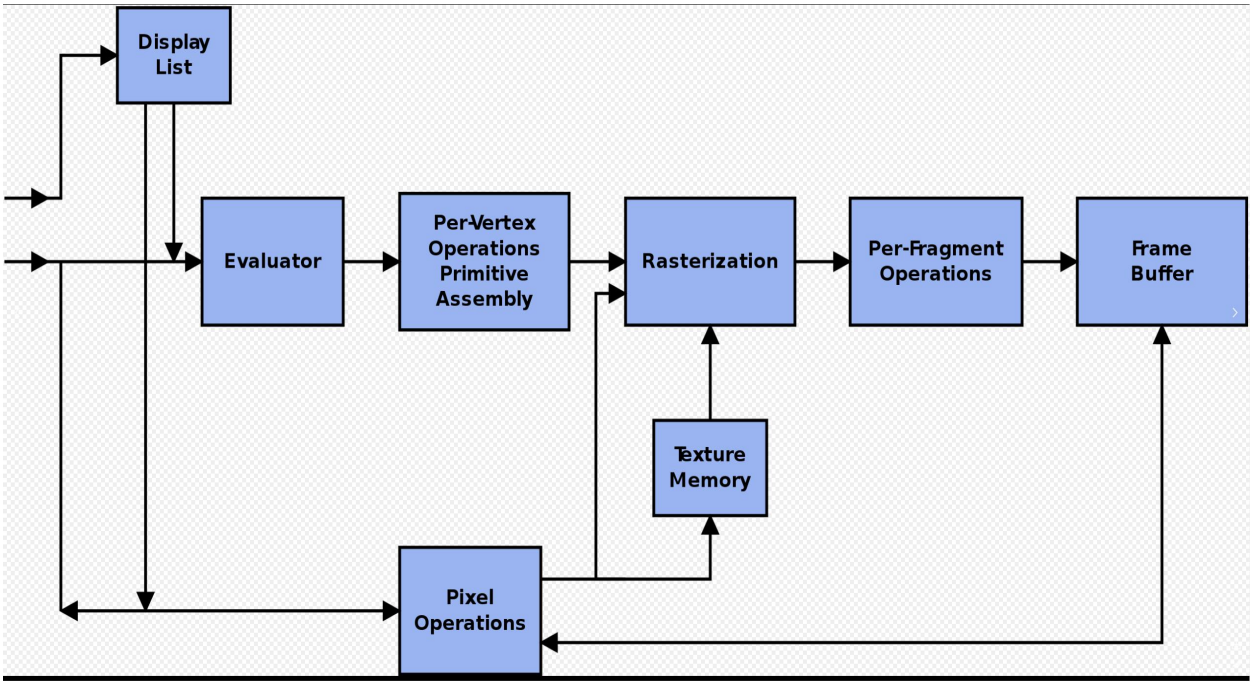
Активниот развој на OpenGL беше отфрлен во корист на Vulkan, објавен во 2016 година и со кодно име glNext за време на првичниот развој. Во 2017 година, Khronos Group објави дека OpenGL ES нема да има нови верзии[48] и оттогаш се концентрира на развојот на Vulkan и други технологии.[49][50] Како резултат на тоа, одредени способности што ги нудат современите графички процесори, на пр. ray tracing, не се поддржани од OpenGL.

Новите верзии на спецификациите на OpenGL се објавени од Khronos Group, од кои секоја го проширува API за поддршка на различни нови функции. Деталите за секоја верзија се одлучуваат со консензус помеѓу членовите на Групацијата, вклучувајќи ги производителите на графички картички, дизајнерите на оперативни системи и општите технолошки компании како што се Mozilla и Google.[51]

Покрај карактеристиките што ги бара основниот API, продавачите на графички процесорски единици (GPU) може да обезбедат дополнителна функционалност во форма на екстензии. Екстензии може да воведат нови функции, нови константи и може да отстранат ограничувањата на постоечките функции на OpenGL. Продавачите можат да користат екстензии за да ги изложат сопствените API без да им треба поддршка од други продавачи или од групата Khronos како целина, што во голема мера ја зголемува флексибилноста на OpenGL. Сите екстензии се собрани во и дефинирани од страна на регистарот OpenGL.[52] Шејдерите во OpenGL се пишуваат во OpenGL Shading Language (GLSL).

---

[48] „The Future of OpenGL (forum discussion)“. Групацијата Хронос. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/OpenGL
[49] „Khronos News Archives“. Групацијата Хронос. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/OpenGL
[50] „Khronos Blog“. Групацијата Хронос. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/OpenGL
[51] „Khronos Membership Overview and FAQ“. Khronos.org. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/OpenGL
[52] „Khronos OpenGL Registry“. Групацијата Хронос. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/OpenGL




*Слика 10. Илустрација на процесот на OpenGL графичкиот pipeline*

**Vulkan**

Vulkan е API на ниско ниво и отворен стандард за 3Д графика и компјутери.[53][54][55] Наменет е да ги реши недостатоците на OpenGL и да им овозможи на програмерите поголема контрола врз графичкиот процесор. Дизајниран е да поддржува широк спектар на графички процесори, процесори, процесори со повеќе јадра и оперативни системи.

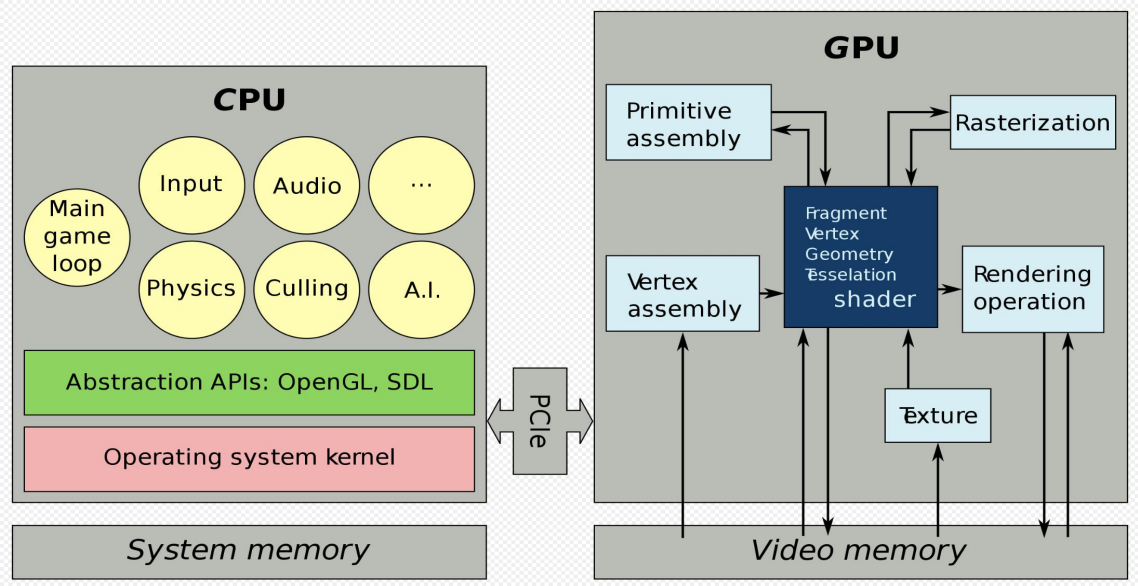
Vulkan цели на 3Д-графички апликации во реално време со високи перформанси, како што се видео игри и интерактивни медиуми и високо паралелизирани компјутери. Vulkan е наменет да понуди повисоки перформанси и поефикасно користење на процесорот и графичкиот процесор во споредба со постарите OpenGL и Direct3D 11 API. Тоа го прави со обезбедување API на значително пониско ниво за апликацијата од постарите API, што поблиску наликува на тоа како функционираат модерните графички процесори.

[53] „Vulkan“. NVIDIA Developer. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Vulkan
[54] „Vulkan®“. GPUOpen. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Vulkan
[55] „Developing 3D Graphics with Vulkan on Mobile“. Мрежа за програмери на Qualcomm. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Vulkan

Vulkan е споредлив со Metal API на Apple и Direct3D 12 на Microsoft. Покрај неговата помала употреба на процесорот, Vulkan е дизајниран да им овозможи на програмерите подобро да ја дистрибуираат работата помеѓу повеќе јадра на процесорот.[56]

Vulkan обезбедува единствен API и за десктоп и за мобилни графички уреди, додека претходно тие беа поделени помеѓу OpenGL и OpenGL ES соодветно.

Vulkan е достапен на повеќе модерни оперативни системи. Како OpenGL, и за разлика од Direct3D 12, Vulkan API не е заклучен на еден оперативен систем или формат на уред. Vulkan работи природно на повеќето модерни оперативни системи.




*Слика 11. OpenGL и Vulkan се API за рендерирање. Во двата случаи, графичкиот процесор извршува шејдери, додека процесорот извршува сè друго*

**Metal**

Metal е API за графичкиот процесор на ниско ниво создаден од Apple. Metal комбинира функции слични на OpenGL и OpenCL во едно API. Наменет е да ги подобри перформансите нудејќи пристап на ниско ниво до хардверот на графичкиот процесор за апликации на iOS, iPadOS, macOS и tvOS. Може да се спореди со API на ниско ниво на други платформи како што се Vulkan и DirectX 12.[57]

Metal има за цел да обезбеди низок пристап до графичкиот процесор. Наредбите се кодирани претходно и потоа се доставуваат до графичкиот процесор за асинхроно

[56] Хрушка, Џоел. „Next-generation Vulkan API could be Valve's killer advantage in battling Microsoft“. ExtremeTech. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Vulkan
[57] https://en.wikipedia.org/wiki/Metal_(API)

извршување. Апликацијата контролира кога да чека да заврши извршувањето, со што им овозможува на развивачите на апликации да ја зголемат пропусната моќ со кодирање на други команди додека командите се извршуваат на графичкиот процесор или да заштедат енергија со експлицитно чекање за завршување на извршувањето на графичкиот процесор. Дополнително, кодирањето на командите е независно од процесорот, така што апликациите можат независно да ги кодираат командите на секоја нишка на процесорот. И на крај, состојбите на рендерирање се однапред пресметани, овозможувајќи му на двигателот на графичкиот процесор однапред да знае како да го конфигурира и оптимизира цевководот, односно pipeline за рендерирање пред извршување на командата.[58]

Metal ги подобрува можностите на програмирањето на GPGPU со користење на *compute shaders*. Metal користи специфичен јазик за засенчување базиран на C++14, имплементиран со помош на Clang и LLVM.[59] Шејдерите во Metal се пишуваат во Metal Shading Language (MSL).

Metal им овозможува на развивачите на апликации да креираат Metal ресурси како што се бафери, текстури, итн. Ресурсите може да се распределат на процесорот, графичкиот процесор или обете и обезбедува можности за ажурирање и синхронизирање на доделените ресурси.

**GDI/GDI+ (Graphics Device Interface)**

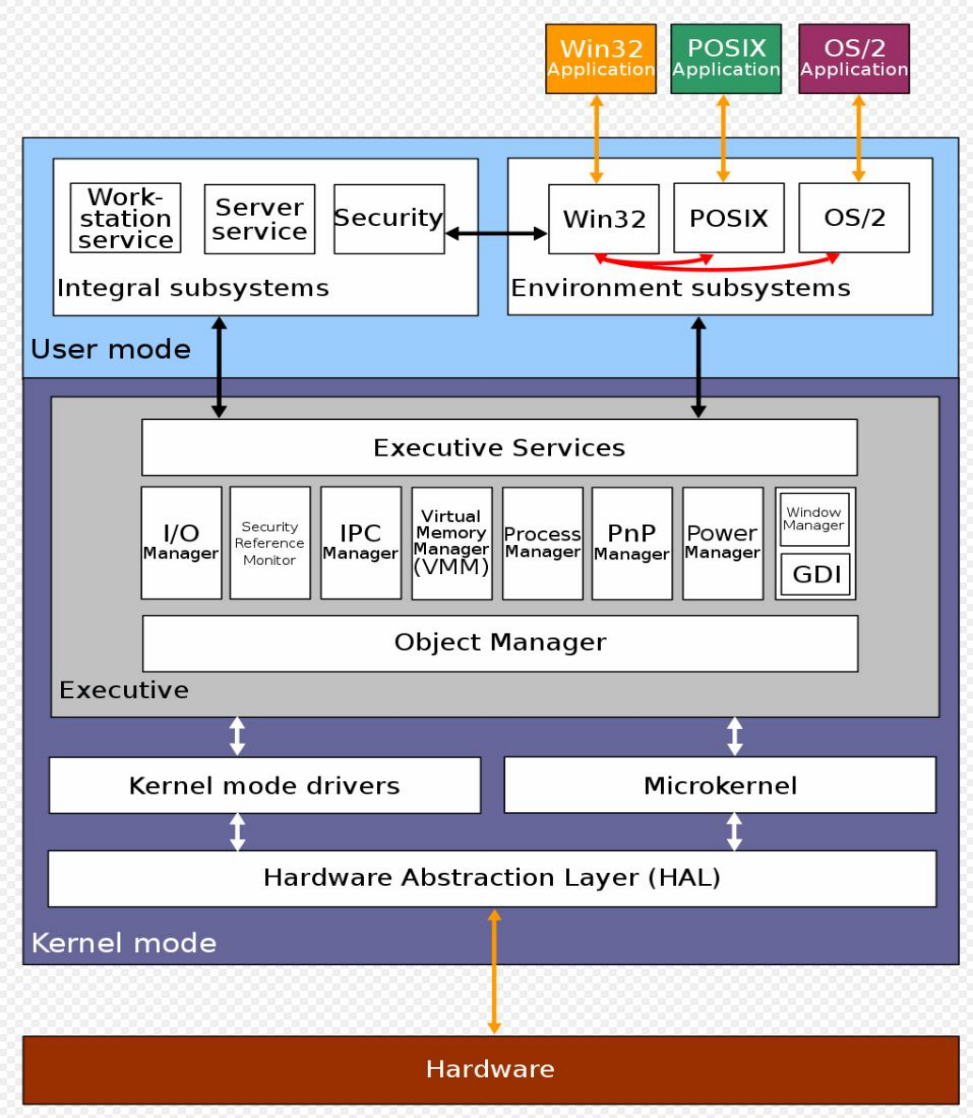
GDI е наследена компонента на Microsoft Windows одговорна за претставување на графички објекти и нивно пренесување на излезни уреди како што се монитори и печатачи. Заменет беше со DirectDraw API, а подоцна и Direct2D API. Апликациите за Windows го користат Windows API за интеракција со GDI, за задачи како што се цртање линии и криви, рендерирање фонтови и ракување со палети и бои. Потсистемот Windows USER користи GDI за прикажување на таквите UI елементи како прозорски рамки и менија. Други системи имаат компоненти кои се слични на GDI; на пример: Mac OS има QuickDraw, а Linux има X Window System.[60]

---

[58] „Setting Up a Command Structure“. Apple Inc. - https://en.wikipedia.org/wiki/Metal_(API)
[59] „Metal Shading Language Guide“. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Metal_(API)
[60] https://en.wikipedia.org/wiki/Graphics_Device_Interface

Најзначајните предности на GDI во однос на подиректните методи за пристап до хардверот се можеби неговите способности за скалирање и неговата апстрактна репрезентација на целните уреди. Користејќи го GDI, можно е да се црта на повеќе уреди, како што се екран и печатач, и да се очекува соодветна репродукција на слика во секој случај.[61]

GDI е релативно тешко да се користи за напредна анимација, нема идеја за синхронизирање со поединечни видео frames во видео картичката и нема хардверска растеризација за 3Д. Современите игри и модерни апликации обично користат DirectX, Vulkan или OpenGL.




[61] https://en.wikipedia.org/wiki/Graphics_Device_Interface

*Слика 12. GDI во архитектурата на Windows NT*[62]

Во GDI, контекстот на уредот (DC, Device Context) ги дефинира атрибутите на текстот и сликите за излезниот уред (пр. екран или печатач). GDI го одржува вистинскиот контекст. За генерирање на излезот е потребен "handle" за контекстот на уредот (HDC, Handle to Device Context).

**DirectX (Direct3D)**

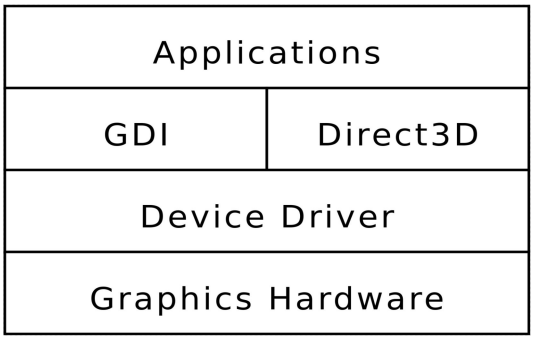
Direct3D е програмски интерфејс за графички апликации (API) за Microsoft Windows. Како дел од DirectX, Direct3D се користи за прикажување на тридимензионална графика во апликации каде што перформансите се важни, како што се игрите. Direct3D користи хардверско забрзување доколку е достапно на графичката картичка, овозможувајќи хардверско забрзување на целиот графички pipeline за 3Д рендерирање или дури и само делумно забрзување. Шејдерите во Direct3D се пишуваат во High-Level Shader Language (HLSL). Direct3D ги изложува напредните графички способности на хардверот за 3Д графика, вклучително и Z-buffering[63], W-buffering[64], баферирање на матрици, просторно антиалиасирање, мешање алфа, мешање бои, mipmapping, мешање текстура,[6566] отсекување, разбивање, атмосферски ефекти, мапирање на текстура со точно перспектива, програмабилни HLSL шејдери[67] и ефекти[68]. Интеграцијата со другите DirectX технологии му овозможува на Direct3D да испорача такви функции како видео мапирање, хардверско 3Д прикажување во рамнини на 2Д преклопување, па дури и sprites, обезбедувајќи употреба на 2Д и 3Д графика во интерактивни медиумски врски.

Direct3D е подсистемска компонента на Microsoft DirectX API. Целта на Direct3D е да ја апстрахира комуникацијата помеѓу графичката апликација и драјверите за графички хардвер. Претставен како тенок апстрактен слој на ниво споредливо со GDI и содржи бројни карактеристики што му недостигаат на GDI.

---

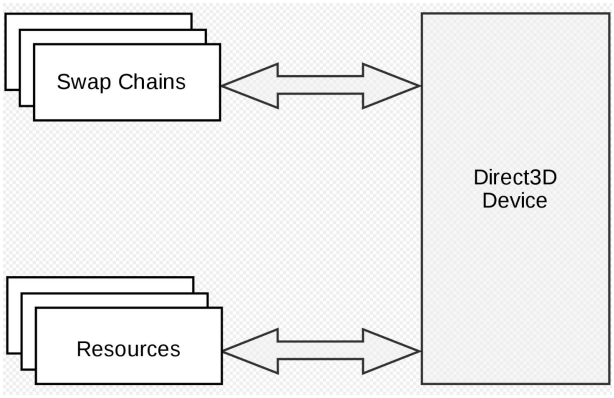
[62] https://en.wikipedia.org/wiki/Graphics_Device_Interface
[63] „Configuring Depth-Stencil Functionality". Мајкрософт. - https://en.wikipedia.org/wiki/Direct3D
[64] „RenderStateManager.UseWBuffer". Мајкрософт. - https://en.wikipedia.org/wiki/Direct3D
[65] „Texture Blending (Direct3D 9)". Мајкрософт. - https://en.wikipedia.org/wiki/Direct3D
[66] „Geometry Blending (Direct3D 9)". Мајкрософт. - https://en.wikipedia.org/wiki/Direct3D
[67] „HLSL". Мајкрософт. - https://en.wikipedia.org/wiki/Direct3D
[68] „Effects (Direct3D 11)". Мајкрософт. - https://en.wikipedia.org/wiki/Direct3D

Direct3D е графички API на непосреден режим. Обезбедува интерфејс на ниско ниво за секоја 3Д функција на видео картичката (трансформации, сечење, осветлување, материјали, текстури, баферирање на длабочини, итн.).




*Слика 13. Апстрактен слој помеѓу GDI и Direct3D*

Непосредниот режим на Direct3D претставува три главни апстракции: уреди, ресурси и синџири за размена (Swap Chains). Уредите се одговорни за рендерирање на 3Д сцената. Тие обезбедуваат интерфејс со различни можности за рендерирање.




*Слика 14. Direct3D уред, неговите ресурси и синџири за размена (Swap Chains)*

На пример, моно уредот обезбедува бело и црно прикажување, додека RGB уредот се прикажува во боја. Постојат четири типа на уреди:

1. Уред HAL (хардверски апстракционен слој): За уреди кои поддржуваат хардверско забрзување.
2. Референтен уред: Симулира нови функции кои сè уште не се достапни во хардверот. Неопходно е да се инсталира Direct3D SDK за да се користи овој тип на уред.

3. Null-Reference уред: не прави ништо. Овој уред се користи кога SDK не е инсталиран и кога се бара референтен уред.
4. Софтверски уред што може да се приклучи: врши рендерирање на софтвер. Овој уред беше претставен со DirectX 9.0.[69]

Секој уред содржи најмалку еден синџир за размена. Разменувачкиот синџир се состои од една или повеќе задни "buffer" површини. Рендерирањето се случува во задниот "buffer".

Microsoft Direct3D 11 API дефинира процес за претворање на група темиња, текстури, бафери и состојби во слика на екранот. Овој процес е опишан како цевковод, односно pipeline, за рендерирање со неколку различни фази. Различните фази на Direct3D 11 се[70]:

1. Input-Assembler[71]: Чита темиња од "buffer" предаден од корисникот и истите темиња ги предава во следните фази.
2. Vertex Shader[72]: Врши операции на едно теме во исто време, како што се трансформации или осветлување.
3. Hull-Shader[73]: Врши операции на множества контролни точки на закрпи и генерира дополнителни податоци познати како константи на закрпи.
4. Tessellator[74]: Ја поделува геометријата за да создаде претстави од повисок ред на трупот.
5. Domain-Shader[75]: Врши операции на излезни темиња од фазата на теселација, на ист начин како и темето засенчување.

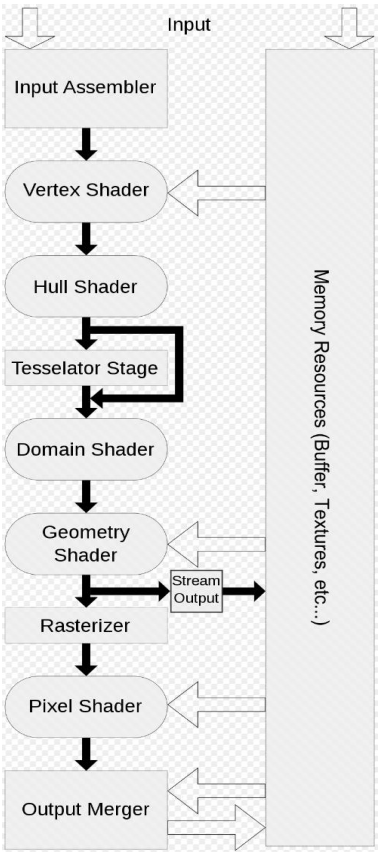
---

[69] „Software Rasterizer for DirectX 9.0 SDK". Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Direct3D
[70] „Graphics Pipeline". Direct3D 11 графика. Microsoft Docs. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Direct3D
[71] „Input-Assembler Stage". Direct3D 11 графика. Microsoft Docs. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Direct3D
[72] „Vertex Shader Stage". Direct3D 11 графика. Microsoft Docs. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Direct3D
[73] „Tessellation Stages". Direct3D 11 графика. Microsoft Docs. Hull-Shader Stage. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Direct3D
[74] „Tessellation Stages". Direct3D 11 графика. Microsoft Docs. Теселатор сцена. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Direct3D
[75] „Tessellation Stages". Direct3D 11 графика. Microsoft Docs. Фаза на Domain-Shader. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Direct3D

6. Geometry Shader[76]: Обработува цели примитиви како што се триаголници, точки или линии. Со оглед на примитивот, оваа фаза го отфрла или генерира еден или повеќе нови примитиви.
7. Stream-Output[77]: Може да ги запише резултатите од претходната фаза во меморија. Корисно за рециркулирање на податоците назад во цевководот, односно pipeline.
8. Rasterizer[7879]: Ги конвертира примитивите во пиксели, внесувајќи ги овие пиксели во шејдерот на пиксели. Растеризаторот може да извршува и други задачи, како што е отсекување на она што не е видливо или интерполирање на податоците од темето во податоци по пиксел.
9. Pixel Shader[80]: Ја одредува конечната боја на пикселот што треба да се запише на целта за прикажување и исто така може да пресмета вредност за длабочина што треба да се запише во баферот за длабочина.
10. Output-Merger[81]: Спојува различни типови излезни податоци (вредности на шејдер на пиксели, мешање на алфа, длабочина, итн.) за да се изгради конечниот резултат.

---

[76] „Geometry Shader Stage“. Direct3D 11 графика. Microsoft Docs. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Direct3D
[77] „Stream-Output Stage“. Direct3D 11 графика. Microsoft Docs. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Direct3D
[78] „RenderStates“. DirectX 9.0 за управуван код. Microsoft Docs. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Direct3D
[79] „Rasterizer Stage“. Direct3D 11 графика. Microsoft Docs. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Direct3D
[80] „Pixel Shader Stage“. Direct3D 11 графика. Microsoft Docs. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Direct3D
[81] „Output-Merger Stage“. Direct3D 11 графика. Microsoft Docs. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Direct3D




*Слика 15. Direct3D 11 процес на графички цевковод, односно pipeline*

Фазите на цевководот илустрирани со изобличен правоаголник се целосно програмибилни. Апликацијата обезбедува програма за шејдер која ги опишува точните операции што треба да се завршат за таа фаза. Многу фази се опционални и можат целосно да се оневозможат.

**ANGLE**

ANGLE (Almost Native Graphics Layer Engine) е слој за апстракција на графичкиот "engine" со отворен код, меѓуплатформски, развиен од Google.[82] ANGLE ги преведува повиците направени преку OpenGL ES 2/3 во повици кон DirectX 9, 11, OpenGL или Vulkan API.[83][84][85][86] Преносliва верзија на OpenGL, но со ограничувања на стандардот

[82] „Google Announces ANGLE – Almost Native Graphics Layer Engine – so OpenGL ES can run over DirectX 9 – khronos.org news ". Khronos.org. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/ANGLE_(software)
[83] „ANGLE". Skia. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/ANGLE_(software)
[84] „Google's Angle brings OpenGL to Windows". I-programmer.info. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/ANGLE_(software)
[85] „Chromium gets GPU acceleration – The H Open: News and Features". H-online.com.

OpenGL ES.[8788]

API е главно дизајнирано за да појави компатибилност со OpenGL, со високи перформанси, на MS Windows и со веб-прелистувачи како што е Chromium преку преведување на повиците на OpenGL во Direct3D, кој има многу подобра поддршка за драјвери на Windows системите. На Windows, постојат два "backend" рендерери за ANGLE: најстариот користи Direct3D 9.0c, додека поновиот користи Direct3D 11.[89]

ANGLE моментално се користи од Chromium (вграден во изворниот код на Blink "engine"), Firefox[90], Edge, WebKit и Qt Framework[91]. Исто така, се користи од Windows 10 за компатибилност со апликации пренесени од Android[92]. Во текот на 2019 година, тимот на Apple придонесе Metal API backend за ANGLE за уредите на Apple да можат да работат на нивниот природен графички API.[93]

**Cairo**

Cairo е графичка библиотека со отворен код која обезбедува API базирана на векторска графика, независен од уред за развивачите на софтвер. Обезбедува примитиви за дводимензионално цртање низ голем број различни backend делови. Cairo користи хардверско забрзување[94] кога е достапно.

Cairo поддржува излез (вклучувајќи растеризација) на голем број различни backend делови, познати како „површини". Поддршката за backends вклучува излез до системот X Window, преку Xlib и XCB, Win32 GDI, OS X Quartz Compositor, OpenGL, локални бафери за слики, PNG-датотеки, PDF, PostScript и SVG-датотеки.

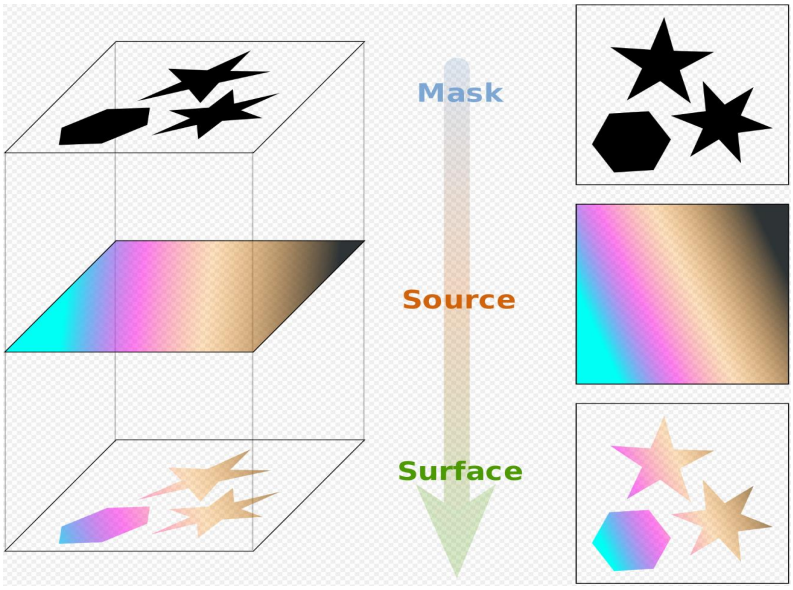
Моделот на цртање во Cairo се потпира на модел со три слоја.

---

Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/ANGLE_(software)
[86] Бриц, Хенри (2010). „Chromium Blog: Introducing the ANGLE Project".
Blog.chromium.org. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/ANGLE_(software)
[87] „angle/angle - Git at Google". chromium.googlesource.com. Преземено на 2024-06-06. -
https://en.wikipedia.org/wiki/ANGLE_(software)
[88] ANGLE: OpenGL на Vulkan. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/ANGLE_(software)
[89] „angleproject – ANGLE: Almost Native Graphics Layer Engine – Google Project Hosting ". Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/ANGLE_(software)
[90] „(WebGL) How to Enable Native OpenGL in your Browser (Windows)". Geeks3D. Преземено на 2024-06-06. -
https://en.wikipedia.org/wiki/ANGLE_(software)
[91] „Graphics on Windows from a different angle | Qt Blog". Qt. Преземено на 2024-06-06. -
https://en.wikipedia.org/wiki/ANGLE_(software)
[92] „ANGLE: Running OpenGL ES 2.0 Graphics Code on Windows". channel9.msdn.com. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/ANGLE_(software)
[93] „WebGL 2.0 Achieves Pervasive Support from all Major Web Browsers ". Групата Хронос. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/ANGLE_(software)
[94] „Cairo homepage". Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Cairo_(graphics)

Секој процес на цртање се одвива во три чекори:

1. Прво се креира „маска”, која вклучува еден или повеќе векторски примитиви или форми, т.е. кругови, квадрати, фонтови, криви, итн.
2. Потоа мора да се дефинира изворот, кој може да биде боја, градиент на боја, битмапа или некоја векторска графика, а од обоените делови на овој извор се прави рез со помош на горедефинираната маска.
3. Конечно, резултатот се пренесува на дестинацијата или површината, која е обезбедена од задниот дел за излезот.




*Слика 16. Моделот на цртање во Каиро*[95]

[95] https://en.wikipedia.org/wiki/Cairo_(graphics)

```c
#include <cairo-svg.h>
#include <stdio.h>

int main(int argc, char **argv) {
    cairo_surface_t *surface = cairo_svg_surface_create("Cairo_example.svg", 100.0, 100.0);
    cairo_t *cr = cairo_create(surface);

    /* Draw the squares in the background */
    for (int x = 0; x < 10; ++x)
       for (int y = 0; y < 10; ++y)
           cairo_rectangle(cr, x * 10.0, y * 10.0, 5, 5);

    cairo_pattern_t *pattern = cairo_pattern_create_radial(50, 50, 5, 50, 50, 50);
    cairo_pattern_add_color_stop_rgb(pattern, 0, 0.75, 0.15, 0.99);
    cairo_pattern_add_color_stop_rgb(pattern, 0.9, 1, 1, 1);

    cairo_set_source(cr, pattern);
    cairo_fill(cr);

    /* Writing in the foreground */
    cairo_set_font_size (cr, 15);
    cairo_select_font_face (cr, "Georgia", CAIRO_FONT_SLANT_NORMAL, CAIRO_FONT_WEIGHT_BOLD);
    cairo_set_source_rgb (cr, 0, 0, 0);

    cairo_move_to(cr, 10, 25);
    cairo_show_text(cr, "Hallo");

    cairo_move_to(cr, 10, 75);
    cairo_show_text(cr, "Wikipedia!");

    cairo_destroy(cr);
    cairo_surface_destroy(surface);
}
```




*Слика 17. Пример во C/C++ како се користи Cairo. Долу десно е резултатот генериран од самиот код*[96]

**Skia**

Skia е 2Д графичка библиотека со отворен код напишана во C++. Skia ги апстрахира графичките API специфични за платформата[97]. Skia Inc. првично ја разви библиотеката; Google ги купи во 2005 година[98], и потоа го објави софтверот како отворен код под New BSD лиценца во 2008 година.

За да остане мулти-платформа, Skia поддржува неколку backend делови (зависни од платформата). Тие вклучуваат:

- CPU растеризација.
- Излез за преносliв формат на документ (PDF).

[96] https://en.wikipedia.org/wiki/Cairo_(graphics)
[97] „Skia Graphics Library“. skia.org. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Skia_Graphics_Engine
[98] Малик, Ом (2008). „ Google Open Sources Skia Graphics Engine“. gigaom. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Skia_Graphics_Engine

- Рендерирање забрзано со графички процесор, користејќи[99]:
    - ANGLE, кој ги преведува повиците на OpenGL ES во природен графички API на платформата.
    - OpenGL, DirectX, Vulkan и Metal.
- Скалабилна векторска графика (SVG).
- Спецификација за XML хартија (XPS).
- Цртеж (Picture) (за снимање команди).

Skia е библиотека која се фокусира на цртање, наместо да обезбедува свои графички контроли и јазик за опис на интерфејсот, итн.[100]

Библиотеката се користи во Google Chrome, ChromeOS, ChromiumOS, Mozilla Firefox, Mozilla Thunderbird, Android, Firefox OS, Flutter[101], итн.

Графичката библиотека Skia може да се користи за цртање текст, геометрии и слики:

- 3x3 матрици со перспектива.
- antialiasing, транспарентност, филтри.
- шејдери, xfermodes, филтри за маски, патеки.
- Потпикселен текст, односно subpixel text.

***Управување со боја на Skia***

Сите простори во боја со кои работи Skia се опишуваат себеси со тоа како да ги трансформираат боите од тој простор на бои во заеднички простор за боја „поврзување“ наречен XYZ D50. И можеме да заклучиме од истиот опис како да се трансформираме од просторот XYZ D50 назад во оригиналниот простор за бои. XYZ D50 е простор во боја претставен во три димензии како RGB, но XYZ деловите воопшто не се слични на RGB, туку линеарен ремикс на тие канали. Y е најблиску до она што би го замислиле како осветленост, но X и Z се малку поапстрактни, слично како YUV моделот за боја. Делот „D50“ се однесува на белата точка на овој простор, околу 5000 Келвини.[102]

---

[99] „Specialized Builds“. Skia. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Skia_Graphics_Engine
[100] „The Skia 2D Graphics Library From Google - Phoronix“. www.phoronix.com. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Skia_Graphics_Engine
[101] „Skia Documentation“. skia.org. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Skia_Graphics_Engine
[102] https://skia.org/docs/user/color/

Целиот цртеж управуван со боја е поделен на шест дела, три чекори ги поврзуваат изворните бои со тој простор на XYZ D50, а потоа три симетрични чекори се поврзуваат назад од XYZ D50 до одредишниот простор за бои. Некои од овие чекори може да се прескокнат, и да резултира со "no-op", односно тој чекор да нема никакви операции да изврши за конверзија на боја. Понекогаш е возможно целиот процес да резултира со никакви конверзии, кога изворниот простор и просторот за дестинација се исти. Чекорите се следни[103]:

1. Се враќа во состојба пред множењето, односно *unpremultiply* ако изворната боја е претходно веќе помножена - алфата не е вклучена во управувањето со боите и треба да ја поделиме ако е веќе помножена.
2. Се линеаризира изворната боја користејќи ја функцијата за пренос на изворниот простор за боја.
3. Се конвертираат тие што не се претходно помножени, линеарни изворни бои во XYZ D50 гама со множење со 3x3 матрица.
4. Се конвертираат оние бои на XYZ D50 во целната гама со множење повторно со 3x3 матрица.
5. Се шифрираат тие бои користејќи инверзната функцијата за пренос на одредишниот простор за боја.
6. Повторно се помножува со алфа ако дестинацијата е претходно помножена.

***Оптимизации***

Секогаш кога се повикува повик за цртање, се гледа кои од тие чекори навистина треба да се извршат. Секој чекор што е фундаментално не е операција, се прескокнува[104]:

- Се прескокнува чекор 1 ако изворот веќе не е помножен.
- Се прескокнува чекор 2 ако изворот е веќе линеарно кодиран.
- Се прескокнуваат чекор 3 и чекор 4 ако таа единечна споена матрица е идентитет (т.е. просторите за бои на изворот и дестинацијата имаат ист опсег).
- Се прескокнува чекор 5 ако дестинацијата бара линеарно кодирање.

[103] https://skia.org/docs/user/color/
[104] https://skia.org/docs/user/color/

- Се прескокнува чекор 6 ако дестинацијата бара да не биде повторно помножена.

Постои опција од тие основни прескокнувања да се извршат уште некои понапредни оптимизации:

- Ако веќе се прескокнати чекорите 3 и 4, може да се прескокнат и чекорите 2 и 5 во секое време кога функциите за пренос се исти - испраќање бои преку дадена функција за пренос и нејзината инверзна функција не е операција.
- Ако веќе се прескокнати сите чекори од 2 до 5, може да се прескокнат и чекор 1 и чекор 6 ако веќе требало да се направат и двете - нема смисла да се врати во состојба пред множењето и само за да повторно да се помножи.
- Непросирните бои може да се третираат или како во состојба пред множењето или однапред помножени, што и да дозволува за да се прескокнат повеќе чекори.

Сето ова се спојува со импресивно „нема што да се прави“ најчесто. Ако се цртаат непросирни бои во даден простор на бои до дестинација означена со истиот простор во боја, може да се забележи дека може да се прескокнат сите шест чекори. Понекогаш се потребни помалку чекори, понекогаш повеќе. Во принцип, ако треба да се направи конверзија на опсегот, генерално треба да се очекува дека сите средни чекори ќе бидат активни. Чекорите 2 и 5 се убедливо најскапи за пресметување.

***Класи и објекти на Skia***

Сите класи, структури и константи во Skia се дефинираат со префикс “Sk”. Некои клучни класи се[105]:

- SkAutoCanvasRestore - Менаџер на стек за зачувување на платно.
- SkBitmap - дводимензионална растерска пикселна низа.
- SkBlendMode - аритметика на боја на пиксели.
- SkCanvas - контекст на цртање (платно).
- SkColor - кодирање во боја користејќи цели броеви.
- SkFont - стил на текст и фонт.
- SkImage - дводимензионална низа од пиксели за цртање.

[105] https://skia.org/docs/user/api/

- SkImageInfo - димензии и карактеристики на пикселите.
- SkIPoint - две целобројни координати.
- SkIRect - цел број правоаголник.
- SkMatrix - 3x3 трансформациона матрица.
- SkPaint - боја, stroke, фонт, ефекти.
- SkPath - низа од поврзани линии и криви.
- SkPicture - низа на команди за цртање.
- SkPixmap - мапа на пиксели: информации за сликата и адреса на пиксели.
- SkPoint - две координати со подвижна запирка.
- SkRRect - заоблен правоаголник со подвижна точка.
- SkRect - правоаголник со подвижна запирка.
- SkRegion - компресирана маска за клипинг.
- SkSurface - дестинација за цртање.
- SkTextBlob - сетови на хиероглифи.
- SkTextBlobBuilder - конструктор за сетови на хиероглифи.

Skia е организирана околу објектот SkCanvas. Тој е домаќин за повиците за „цртање“: drawRect, drawPath, drawText, итн. Секоја од нив има две компоненти: примитивниот што се црта (SkRect, SkPath, итн.) и атрибутите на боја/стил (SkPaint).

Бојата држи голем дел од состојбата опишувајќи како треба да биде нацртан правоаголникот: каква боја е, дали е наполнет (filled) или stroked, како треба да се спои со претходно нацртаното, итн.

Платното има релативно мала состојба. Посочува на цртаните вистински пиксели и одржува куп, односно стек матрици и клипови. Така, во т.н. повик drawRect (rect, paint), тековната матрица на платното може да ги трансформира координатите на правоаголникот (превод, ротација, искривување, перспектива), а моменталниот клип, односно исечок на платното може да ограничи каде на платното ќе биде нацртан правоаголникот, но сите други стилски атрибутите на цртежот се контролирани од бојата[106].

[106] https://skia.org/docs/user/api/

***SkSL и ефекти за време на извршување***

SkSL е јазикот за засенчување на Skia. SkRuntimeEffect е објект на Skia C++ кој може да се користи за креирање објекти SkShader, SkColorFilter и SkBlender со однесување контролирано од SkSL код.

Синтаксата е многу слична на GLSL. Кога се користи SkSL ефекти во апликација која користи Skia, постојат важни разлики (од GLSL) што треба да се запомнат. Повеќето од овие разлики се поради еден основен факт: со јазиците за засенчување на графичкиот процесор, се програмира фаза од цевководот, односно pipeline на графичкиот процесор. Со SkSL, се програмира фаза од цевководот, односно pipeline на Skia.

Особено, Fragment Shader во GLSL го контролира целото однесување на графичкиот процесор помеѓу растеризаторот и хардверот за мешање. Тој шејдер ја врши целата работа за пресметување на бојата, а бојата што ја генерира е токму она што се внесува во фазата на мешање односно блендирање со фиксна функција на pipeline.

Ефектите на SkSL постојат како дел од поголемиот pipeline на Skia. Кога се издава операција за цртање на платно (SkCanvas), Skia (обично) составува еден шејдер на фрагменти од графичкиот процесор за да ја изврши целата потребна работа. Овој шејдер обично вклучува неколку парчиња. На пример, може да вклучува:

- Пресметување дали пиксел паѓа внатре или надвор од обликот што се црта (или на границата, каде што може да примени антиализација).
- Пресметување дали пикселот паѓа внатре или надвор од регионот на клипинг, односно исеченост (повторно, со можна антиализациона логика за граничните пиксели).
- Логика за SkShader на SkPaint. SkShader всушност може да биде дрво на објекти.
- Слична логика за SkColorFilter (што може да биде и дрво, поради SkColorFilters::Compose и SkColorFilters::Blend).
- Код за блендирање (за одредени SkBlendModes или за прилагодено блендирање наведено со SkPaint::setBlender).
- Код за конверзија на простор во боја, како дел од управувањето со бојата на Skia.

Дури и ако SkPaint има комплексно стебло на објекти во полињата SkShader, SkColorFilter или SkBlender, сè уште има само еден шејдер на фрагменти од графичкиот процесор. Секој јазол во тоа дрво создава една функција. Кодот за отсекување и геометријата создаваат функција. Кодот за блендирање може да создаде функција. Целокупниот засенувач на фрагменти потоа ги повикува сите овие функции (кои може да повикаат други функции, на пр. во случај на дрво SkShader)[107].

Ефектите на SkSL придонесуваат функција во Fragment Shader на графичкиот процесор.

***Skia во другите софтвери како Chromium***

Chrome користи Skia за скоро сите графички операции, вклучително и прикажување текст. GDI во најголем дел се користи само за рендерирање на темите на прозорците; новиот код треба да користи Skia[108].

Зашто тимот Chromium ја користи Skia:

- GDI не е доволно опремен за работи како SVG, Canvas и некои од сложените работи на UI.
- Skia е внатрешно развиен од страна на Google и може да се менува за да одговара на нивните потреби.
- Веќе постоело висококвалитетен порт од Webkit/Canvas до Skia.
- GDI+ повеќе не е во развој од Microsoft и е побавен од Skia на повеќето операции, барем на XP. На Vista, се чини дека барем некои GDI+ операции се хардверски забрзани, но софтверот Skia е генерално доволно брз за нивните потреби.
- Презентирањето, односно цртањето на текст во GDI+ е неприфатливо различно од прикажувањето на текстот во GDI и бавно се подига.

Windows обезбедува многу примитивни алатки за градење кориснички интерфејси. Системот обезбедува неколку основни контроли и природни контејнери за прозорци, но изградбата на сопствен кориснички интерфејс е тешка. Бидејќи Chromium тимот сакаат диференцирана естетика за Chromium прелистувачите, морале да изградат framework на

[107] https://skia.org/docs/user/sksl/
[108] https://www.chromium.org/developers/design-documents/graphics-and-skia/

Windows за да го забрзаат развојот на приспособен, односно прилагоден интерфејс. Овој систем се нарекува ***views***[109].

Views е систем за рендерирање што не е многу различен од оној што се користи во WebKit или Gecko за прикажување на веб-страници. Корисничкиот интерфејс е конструиран од дрво на компоненти наречени Views. Овие Views се одговорни за прикажување, распоред и ракување со настани. Секој view во дрвото претставува различна компонента на корисничкиот интерфејс. Аналогот е хиерархиска структура на HTML документ[110].

Во коренот на хиерархијата на View е Widget, кој е "native" прозорец. Native прозорец прима пораки од Windows, ги претвора во нешто што може да го разбере хиерархијата на View системот, а потоа ги пренесува на RootView. RootView потоа започнува со ширење на настанот во хиерархијата View.

Painting и layout се прават на сличен начин. View во дрвото View има свои граници, и затоа, кога ќе биде побарано да црта, црта во платно исечено до неговите граници. Рендерирањето за целото дрво на View се врши во едно платно поставено и во сопственост на Widget кога ќе прими порака за Paint. Самото прикажување се врши со помош на комбинација од повици Skia и GDI[111].

Сепак, неколку контроли на интерфејсот во интерфејсот на Chromium не се прикажуваат со користење на Views. Наместо тоа, тие се native контроли на Windows кои се хостирани во посебен вид View кој знае себе да се прикаже и да ја одреди големината на native Windows контролата. Тие се користат за копчиња, табели, радио копчиња, полиња за избор, текстуални полиња и други такви контроли. Бидејќи користат природни контроли, овие Views исто така не се особено преносливи, освен можеби во API[112].

На почетокот на проектот Chromium, се почнало од градењето на прелистувачот Chromium користејќи native прозорци и користејќи го пристапот за цртање што се користи во многу апликации за Windows. Ова се покажало како незадоволително, бидејќи оригиналните прозорци не поддржуваат транспарентност природно, а справувањето со

[109] https://www.chromium.org/developers/design-documents/chromeviews/
[110] https://www.chromium.org/developers/design-documents/chromeviews/
[111] https://www.chromium.org/developers/design-documents/chromeviews/
[112] https://www.chromium.org/developers/design-documents/chromeviews/

настани бара досадно подкласирање на прозорци. Некои рани елементи на интерфејсот гравитирале кон приспособено цртање и справување со настани (на пр. Autocomplete)[113].

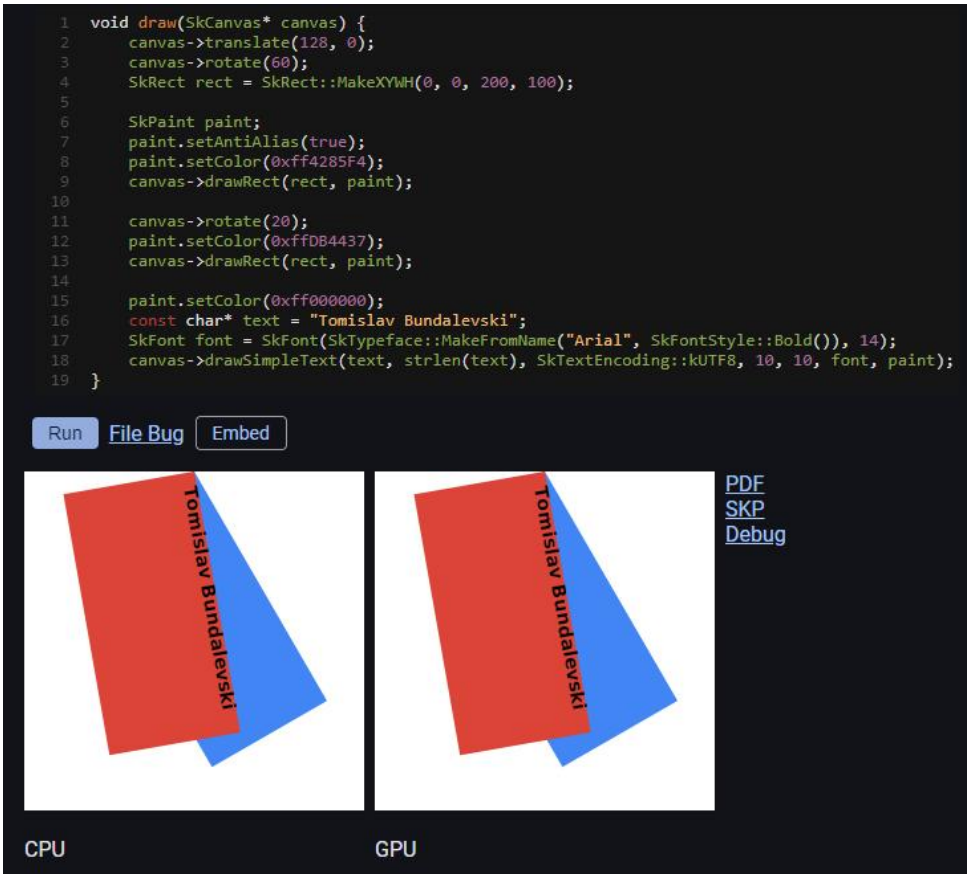
Постоечките комплети за кориснички интерфејси за Windows се слично незадоволувачки, со ограничени комплети на графички контроли, неприродна естетика или незгодни модели на програмирање[114].

Во голема мера, views го олеснуваат градењето сложени сопствени кориснички интерфејси. Сепак, има неколку груби рабови кои може да се подобрат со текот на времето[115]:

- Типовите на настани моментално се повремено проблематични - тие ги пробиваат параметрите на оригиналните пораки на Windows и потоа ги отфрлаат. Понекогаш оваа информација е корисна.
- Некоја ад хок обработка на пораки.
- Мешавина од природни контроли кои не работат правилно додека не се вметнат во хиерархија на View, прикачена на Widget со валиден HWND. Многу од природните контроли имаат API методи кои бараат од нив да постојат во хиерархија на прозорци. Ова значи дека тие не можат целосно да се иницијализираат додека не се вметнат.

***Користење на Skia***

[113] https://www.chromium.org/developers/design-documents/chromeviews/
[114] https://www.chromium.org/developers/design-documents/chromeviews/
[115] https://www.chromium.org/developers/design-documents/chromeviews/

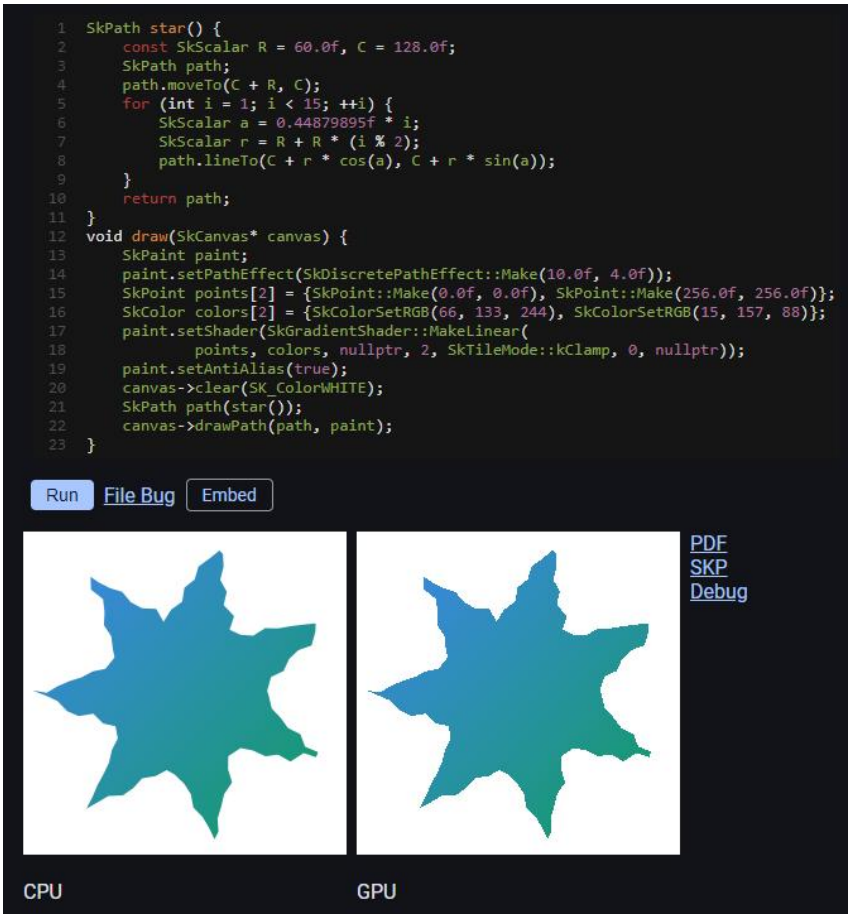



*Слика 18. Пример во C/C++ како се користи Skia. Долу е резултатот генериран од самиот код, соодветно од страна на CPU и GPU*

Мало објаснување за горниот код и слика:

Во горниот дел, се дефинира функција за цртање “draw” со аргумент „платно”, односно canvas која класа се вика “SkCanvas”. Сите класи, структури и константи во Skia се дефинираат со префикс “Sk”. Понатаму, првите 2 команди во canvas го движат и ротираат платното. Кога се прават translate, rotate, scale командите во canvas, тоа во позадина работи со матрица од 3x3 или 4x4. Соодветно, тие имаат ефект на командите за цртање, сечење, итн. Во овој случај, прво се движи платното по x-оската за 128, а потоа се ротира за 60 степени. Ова има ефект на следните команди во canvas да се подвижи геометријата по x-оската за 128, а потоа да ја ротира истата геометрија за 60 степена. Во компјутерска графика, за разлика во математиката, y-оската е превртена наопачки, така што позитивните вредности на y-оската се кога се движиме надолу, додека кога се движиме нагоре по y-оската, се движиме кон негативност. Ова е клучен детаљ, затоа што објаснува зашто ротација од 60 степени како што правиме погоре не ја ротира геометријата како што ја очекуваме од математика. Во случајот, ротација од позитивни

вредности ја ротираат геометријата според часовникот, а не обратно. Првата “drawRect” команда е плавиот правоаголник. Повторно, платното се ротира за уште 20 степени и се црта вториот црвен правоаголник. На крај, колку да се демонстрира и обичен текст, се црта и текстот со црна боја со командата “drawSimpleText”. Како што може да се забележи, текстот е нацртан со последната трансформација на самиот canvas (translateX: 128px, rotate(80deg)).




*Слика 19. Пример во C/C++ како се користи Skia. Долу е резултатот генериран од самиот код, соодветно од страна на CPU и GPU*

# 7. Методологија на истражувањето

**Цели на истражувањето**

Предмет и цел на ова истражување е да се дадат основите на креирање на еден cross-platform framework кој понатаму може да се користи за да се креираат десктоп апликации помеѓу различните оперативни системи, со еден унифициран изглед и функционалност. Од страна на развивачот, многу се олеснува процесот на креирање на една апликација, со оглед на тоа дека исто така се олеснува процесот на дефинирање на елементите кои развивачот планира да ги презентира на корисникот, „стилирање" на елементите, функционалност, итн.

**Задачи на истражувањето**

- Да се направи коректен "build" на Skia и да се поврзе со проектот на кој се планира да се користи библиотеката.
- Да се истражи како самата библиотека функционира во околината, според развивачите (Google), нивните поставени цели и нивното користење на самата библиотека.
- Со тоа се добива перспектива како и зошто некои работи треба да се направат од почеток (пр. да не се користат класите и методите од Windows за креирање на еден <input type="text"> или <input type="date">), туку да се креираат според потребите на имплементацијата на сопствениот framework.
- Со добиената перспектива и експертиза од самите дизајн документи и истражувања на Google и Chromium тимот, може да се креираат имплементации на различни сервиси, алатки и начини на користење на framework од страна на развивачот.
- Да се креираат различни прилагодени контроли наменети за потребите на еден framework, по потреба да се даде можност да се „стилира" понатаму преку различни методи од страна на развивачот, па дури и корисникот.
- Да се креираат различни прилагодени прозори, како дијалог, избирање на датум, context menu, tooltip, итн. наменети за потребите на еден framework, по потреба да се даде можност да се „стилира" понатаму преку различни методи од страна на развивачот, па дури и корисникот.

**Генерална хипотеза**

Генерална хипотеза на истражувањето е да се докаже дека за различни потреби на корисниците, развивачите може да креираат еден прилагоден framework.

**Методи на истражување**

Со цел да се исполнат најважните методолошки пристапи кои се наметнуваат во секој framework, употребени се следните:

- метод на собирање на податоци преку различни извори,
- аналитичко-синтетички метод,
- метод на класификација,
- метод на визуелен приказ,
- компаративен метод,
- метод на конкретизација и генерализација.

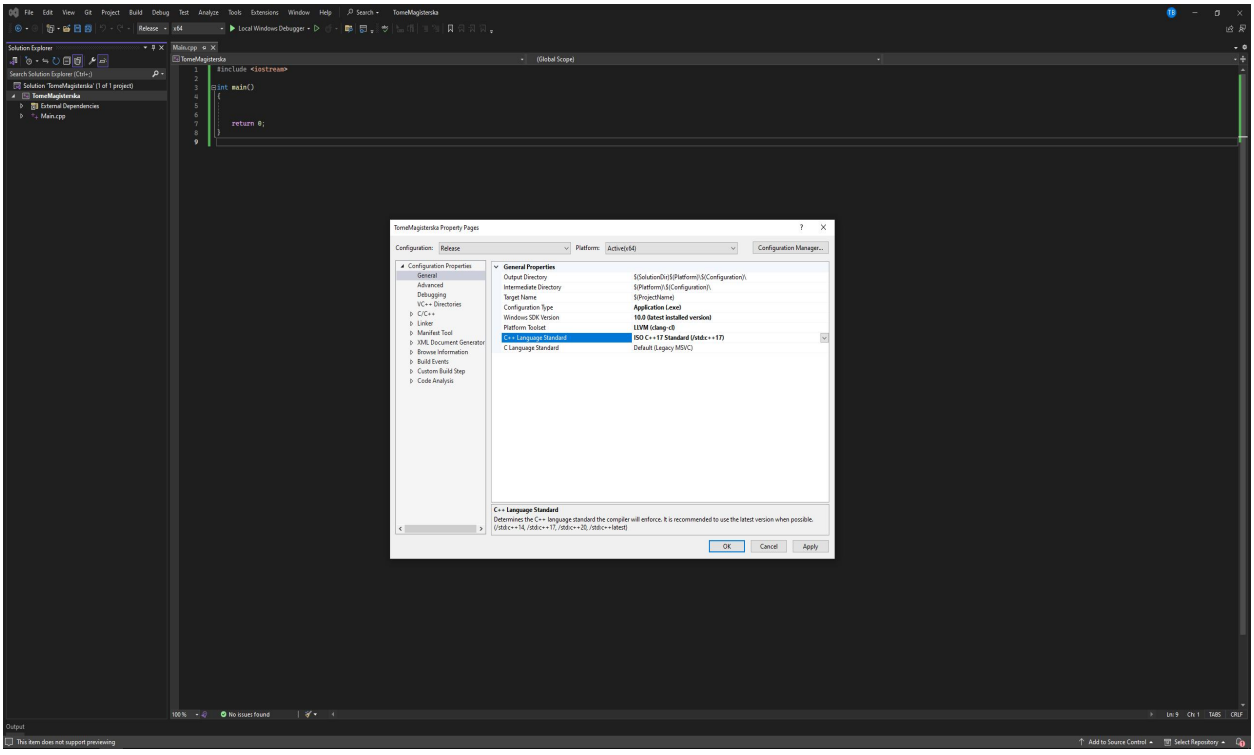
**Оправданост на истражувањето**

Резултатите од истражувањето може да се користат да се креира еден нов framework, за различни намени и цели, според потребите на корисникот. Некои ефекти, начини на приказ, функционалност, начини на функционалност не се возможни, се тешки за имплементација, имаат недостатоци во имплементација или резултираат со високи трошоци од страна на оперативниот систем. Заради тоа, се преоѓа кон креирање на еден нов framework природно за да се надминат споменатите проблеми.

# 8. Практичен дел

Во овој труд, референцирајќи се на погоре изнесените теоретски објаснувања во врска со framework и библиотеки за цртање, ќе се направи обид да се креира една околина каде што ќе може да се демонстрира рендерирање со Skia, и со CPU и со GPU. Понатаму, поради тоа што многу секојдневни и популарни софтвери (како прелистувачи) ја користат библиотеката Skia, ќе бидат споредени резултатите рендерирани/нацртани од познатите софтвери и од имплементацијата презентирана во овој труд. Исто така, ќе може да се спореди колку CPU/GPU/Memory се користи и на крај, колку креираната апликација зафаќа тотално простор на диск, итн.

Прво, се одлучува околината како ќе биде поставена, кој програмски јазик ќе се користи и која верзија, кој компајлер и која верзија, кои библиотеки, итн. Користената библиотека за цртање е Skia, а Skia најдобро се компајлира со C++17 Clang/LLVM. C++ нема стандардизирано ABI за дефинирање на крајниот бинарен излез. Поради тие причини, во овој труд се користат компатибилните или истите верзии на C++ и компајлери. Clang/LLVM може да се инсталира преку Visual Studio Installer. Да се креира таков почетен C++ проект во Visual Studio 2022.



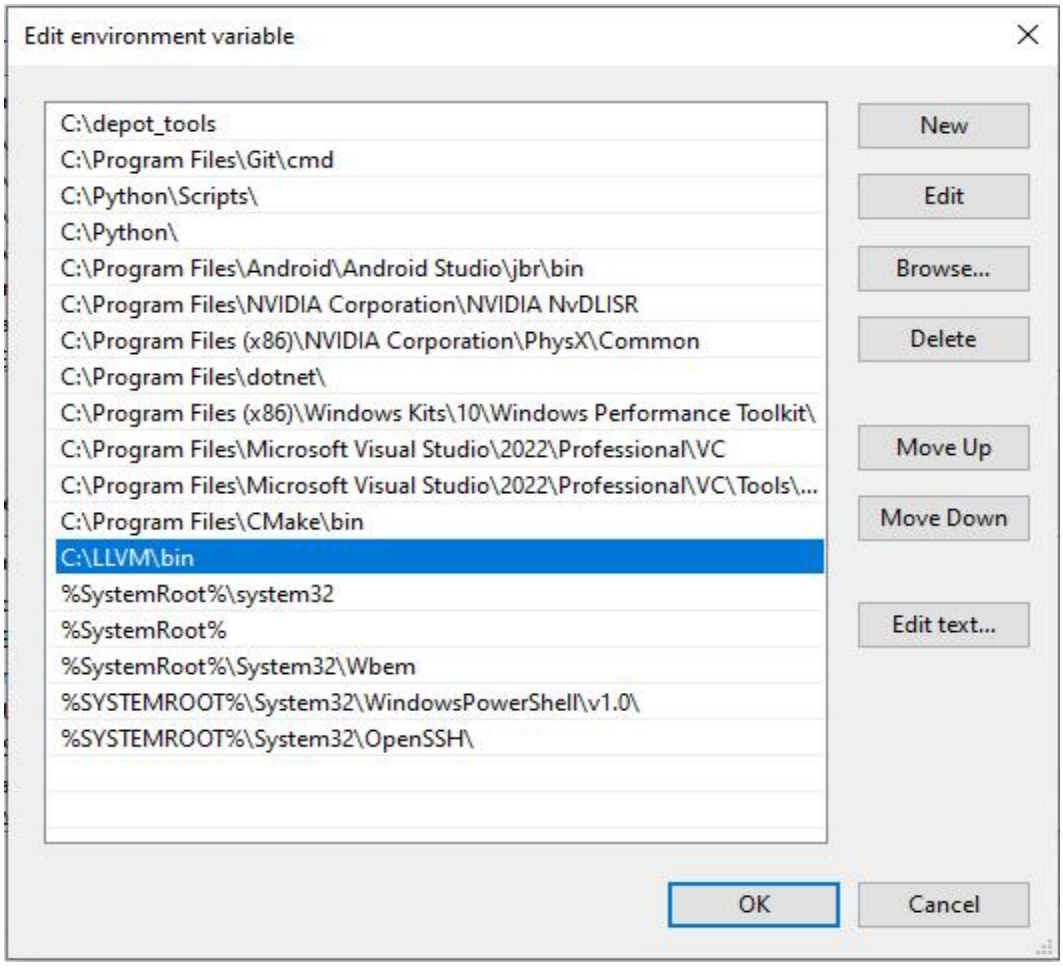
*Слика 20. Креирање на проект во Visual Studio и конфигурирање на C++ верзија и компајлер*

Следно е да се искомпајлира Skia со истите конфигурации како што се направени погоре и потоа статички да се поврзе со проектот со “/MT” компајлерска опција (multi-threading).

Самите инструкции за тоа како се презема Skia, неговите зависни библиотеки, неговите модули, како се конфигурира Skia за да се изврши коректен “Build” е малку комплицирано, но е документирано од самите развивачи на Skia на нивната официјална страна за документација.

Неколку различни софтвери како Git, Python, depot_tools (Google алатка), ninja (build алатка) и самиот компајлер Clang/LLVM се потребни за да се искомпајлира и изврши коректен “build” на Skia.

Во овој труд, Clang/LLVM компајлерот е поврзан, така што на Windows System Environment variables е додаден во променливата “PATH”. Овој компајлер е сличен на тој што се користи и од страна на проектот (иста или слична верзија):

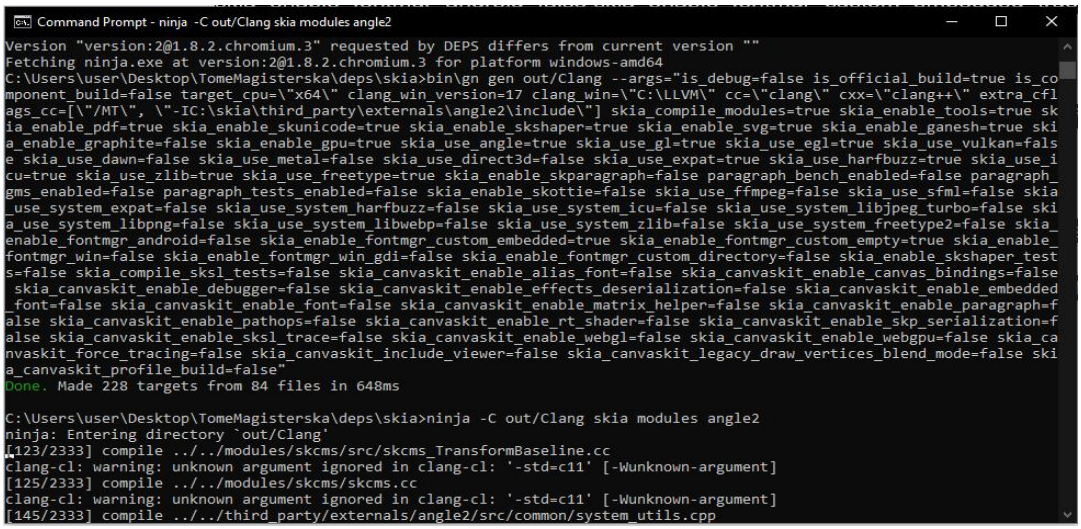



*Слика 21. Поставување на Clang/LLVM во Windows System Environment variables*

Откога ќе се завршат сите побарувања дадени од тимот Skia (инсталација на различни потребни софтвери, итн.), командите за конфигурација и компајлирање на Skia се следните:

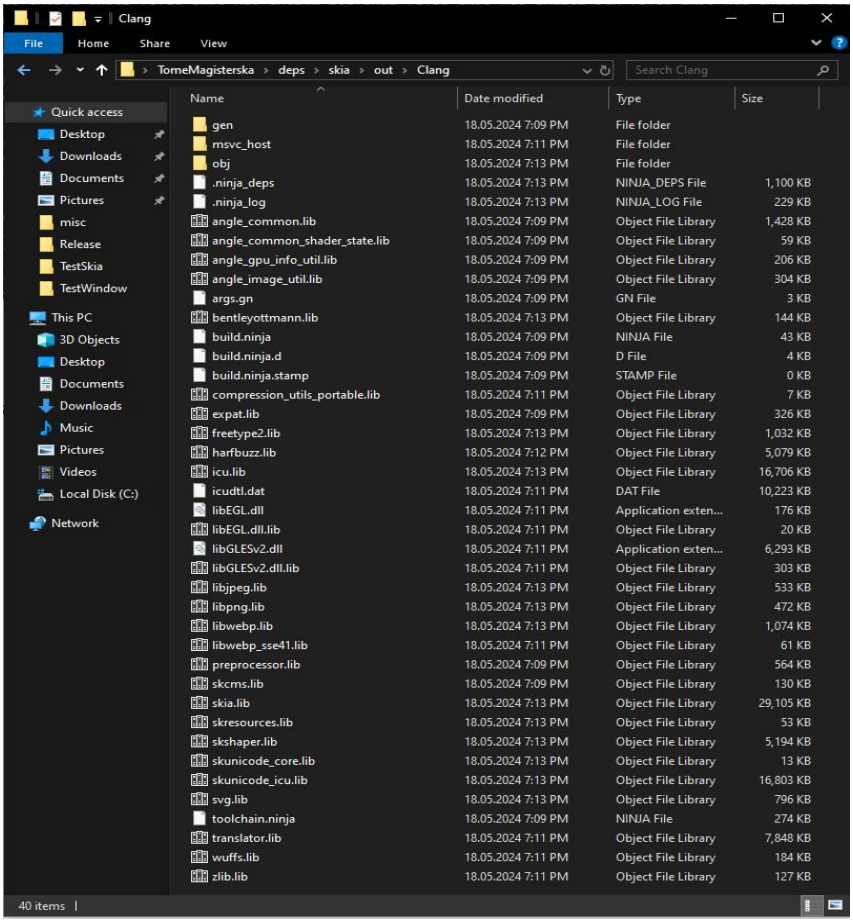
1. git clone https://github.com/google/skia
2. python3 tools/git-sync-deps
3. python3 bin/fetch-ninja
4. **bin\gn gen out/Clang --args="is_debug=false is_official_build=true is_component_build=false target_cpu=\"x64\" clang_win_version=17 clang_win=\"C:\LLVM\" cc=\"clang\" cxx=\"clang++\" extra_cflags_cc=[\"/MT\", \"-IC:\skia\third_party\externals\angle2\include\"] skia_compile_modules=true skia_enable_tools=true skia_enable_pdf=true skia_enable_skunicode=true skia_enable_skshaper=true skia_enable_svg=true skia_enable_ganesh=true skia_enable_graphite=false skia_enable_gpu=true skia_use_angle=true skia_use_gl=true skia_use_egl=true skia_use_vulkan=false skia_use_dawn=false skia_use_metal=false skia_use_direct3d=false skia_use_expat=true skia_use_harfbuzz=true skia_use_icu=true skia_use_zlib=true skia_use_freetype=true skia_enable_skparagraph=false paragraph_bench_enabled=false paragraph_gms_enabled=false paragraph_tests_enabled=false skia_enable_skottie=false skia_use_ffmpeg=false skia_use_sfml=false skia_use_system_expat=false skia_use_system_harfbuzz=false skia_use_system_icu=false skia_use_system_libjpeg_turbo=false skia_use_system_libpng=false skia_use_system_libwebp=false skia_use_system_zlib=false skia_use_system_freetype2=false skia_enable_fontmgr_android=false skia_enable_fontmgr_custom_embedded=true skia_enable_fontmgr_custom_empty=true skia_enable_fontmgr_win=false skia_enable_fontmgr_win_gdi=false skia_enable_fontmgr_custom_directory=false skia_enable_skshaper_tests=false skia_compile_sksl_tests=false skia_canvaskit_enable_alias_font=false skia_canvaskit_enable_canvas_bindings=false skia_canvaskit_enable_debugger=false skia_canvaskit_enable_effects_deserialization=false skia_canvaskit_enable_embedded_font=false skia_canvaskit_enable_font=false skia_canvaskit_enable_matrix_helper=false skia_canvaskit_enable_paragraph=false skia_canvaskit_enable_pathops=false skia_canvaskit_enable_rt_shader=false skia_canvaskit_enable_skp_serialization=false skia_canvaskit_enable_sksl_trace=false skia_canvaskit_enable_webgl=false skia_canvaskit_enable_webgpu=false skia_canvaskit_force_tracing=false skia_canvaskit_include_viewer=false skia_canvaskit_legacy_draw_vertices_blend_mode=false skia_canvaskit_profile_build=false"**

5. ninja -C out/Clang skia modules angle2




*Слика 22. Компајлирање на Skia со ANGLE backend*

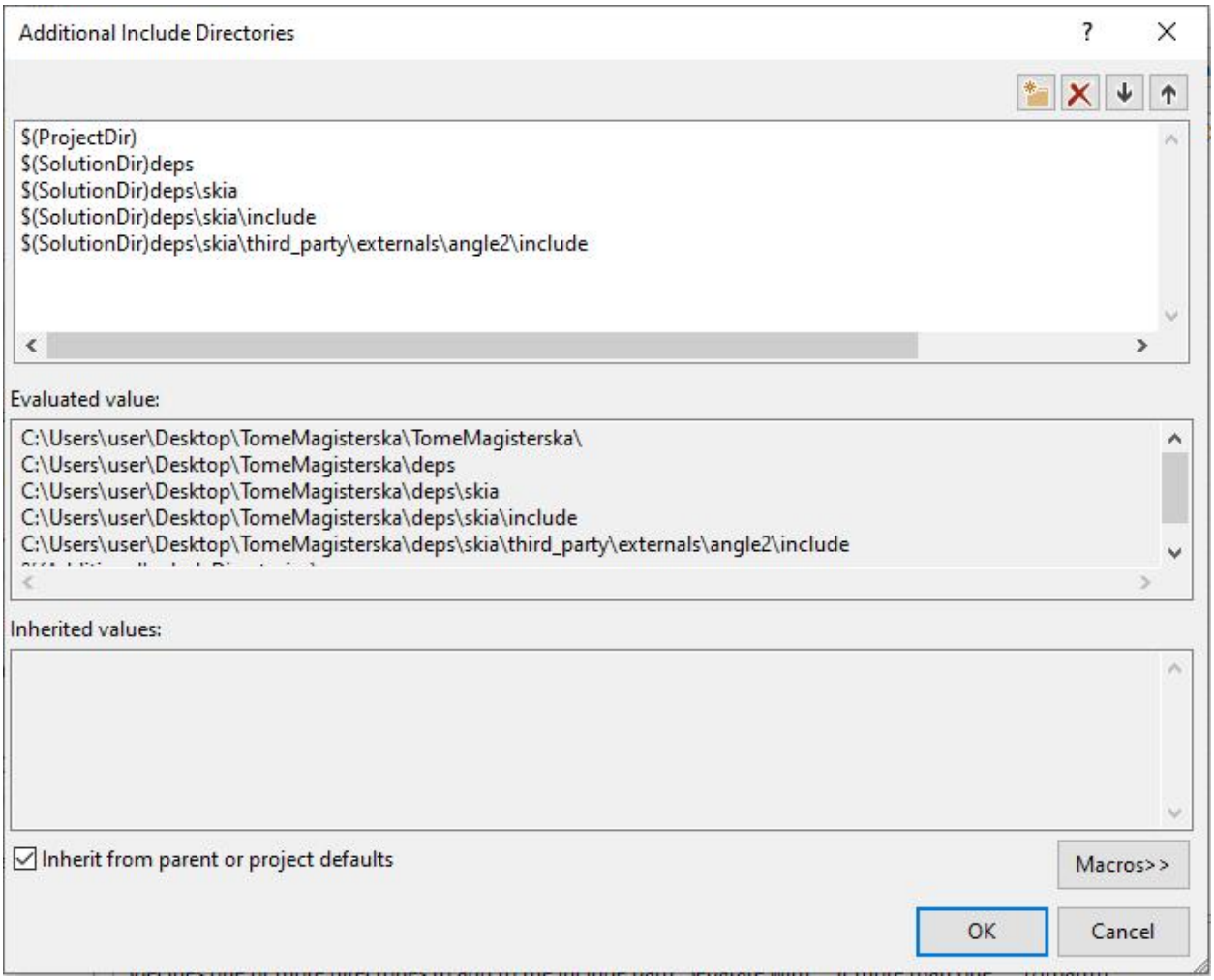
Сите искомпајлирани објекти и библиотеки се наоѓаат во директориумот “skia/out/Clang”.




*Слика 23. Резултатот од компајлирање на Skia со ANGLE backend*

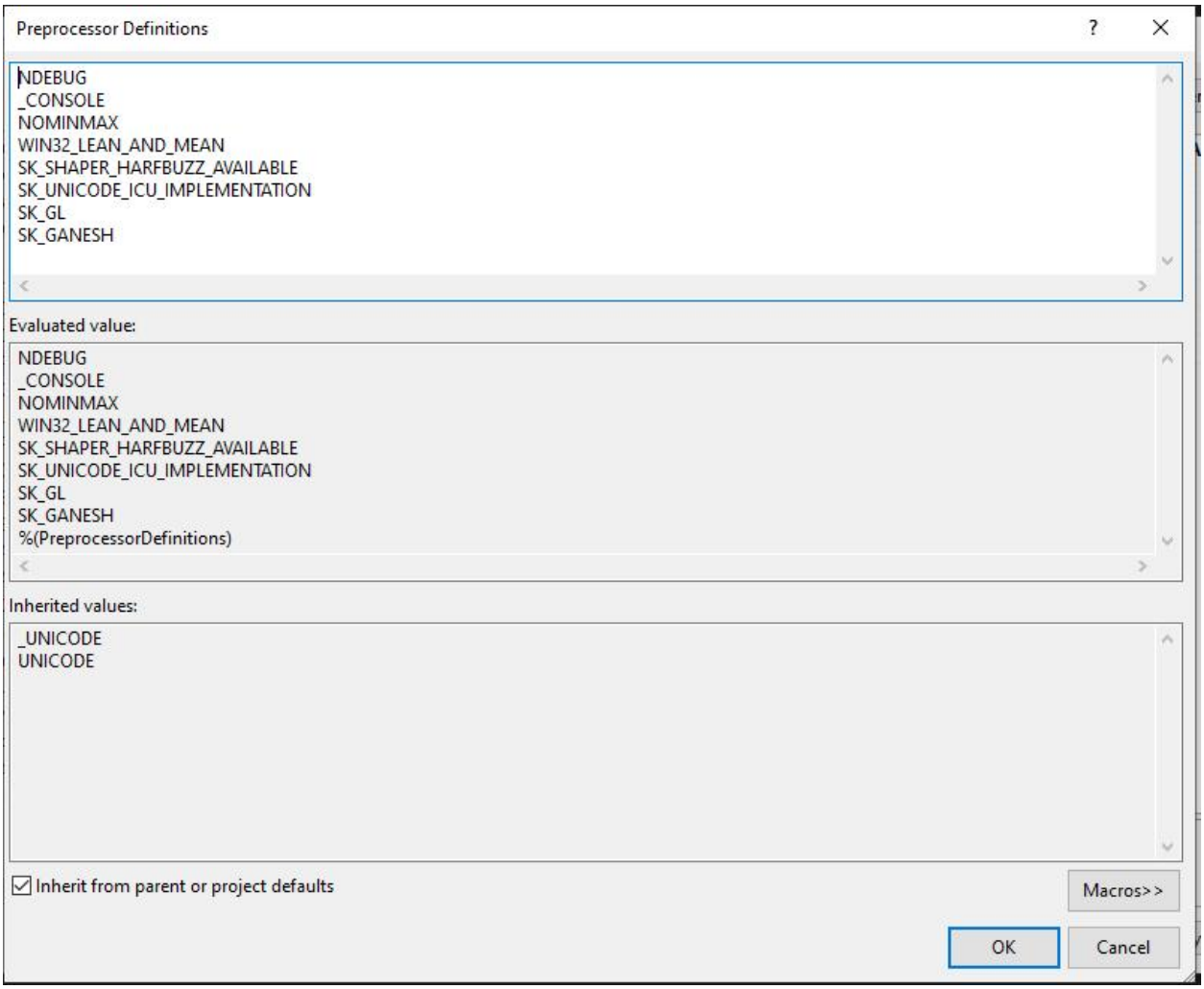
Во проектот треба овие објекти да се референцираат и статички да се поврзат. Тоа го правиме со следните чекори:

1. Во Configuration Properties -> C/C++ -> General се поставуваат патеките каде се наоѓаат хедер односно .h фајловите во Skia. Поради тоа што Skia се наоѓа во нашиот “Solution”, патеката на Skia е TomeMagisterska/deps/skia. Во Additional Include Directories, ги поставуваме следните ставки:
   - $(ProjectDir)
   - $(SolutionDir)deps
   - $(SolutionDir)deps\skia
   - $(SolutionDir)deps\skia\include
   - $(SolutionDir)deps\skia\third_party\externals\angle2\include




*Слика 24. Поставување на Skia include директориуми*

2. Во Configuration Properties -> C/C++ -> Preprocessor се поставуваат следните константи во Preprocessor Definitions:
   - NOMINMAX
   - WIN32_LEAN_AND_MEAN
   - SK_SHAPER_HARFBUZZ_AVAILABLE
   - SK_UNICODE_ICU_IMPLEMENTATION
   - SK_GL
   - SK_GANESH

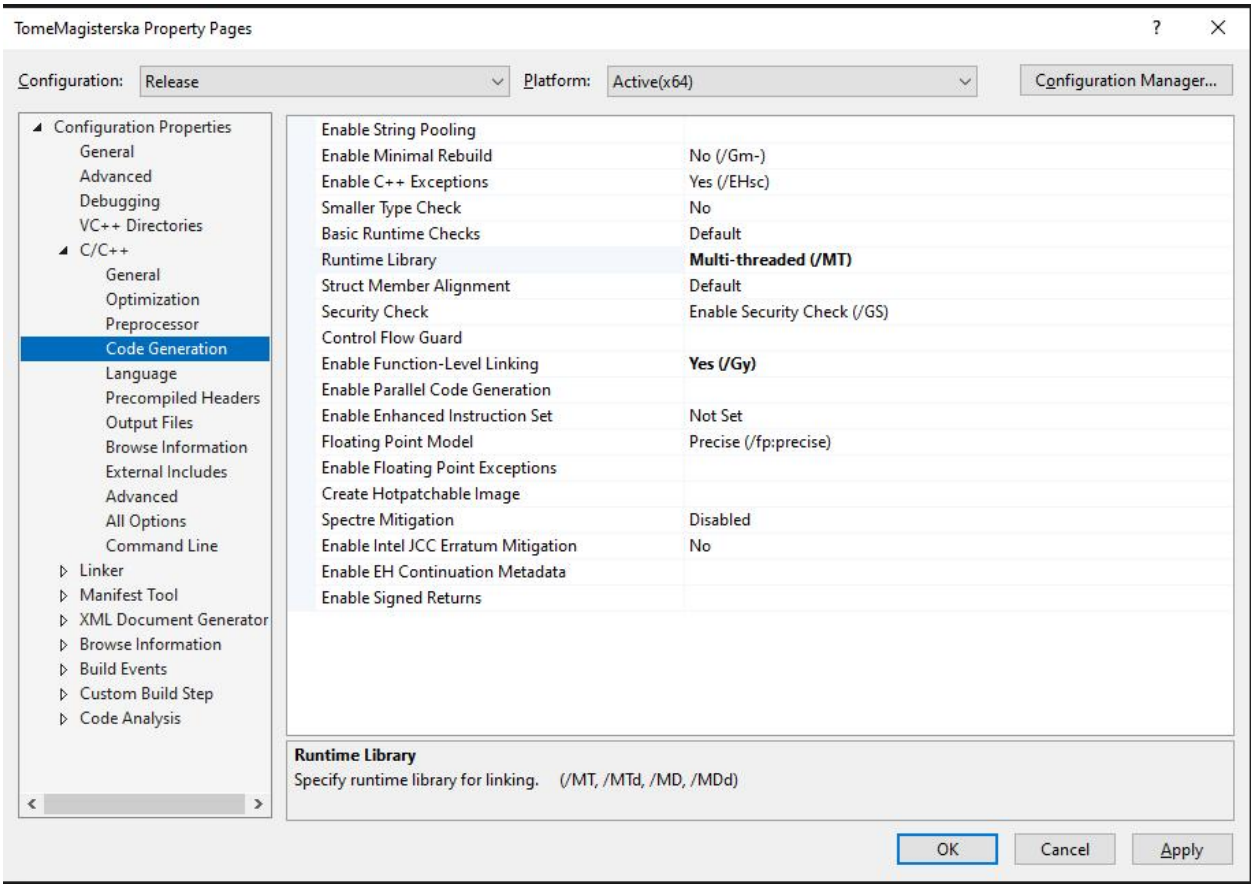



*Слика 25. Поставување на Skia и Windows константи*

NOMINMAX и WIN32_LEAN_AND_MEAN ги поставуваме затоа што имаат конфликт Skia и Windows кога не се редоследно “#include” односно вклучени во еден фајл, доведувајќи до ситуација компајлерот да се буни за некои функции што го користат

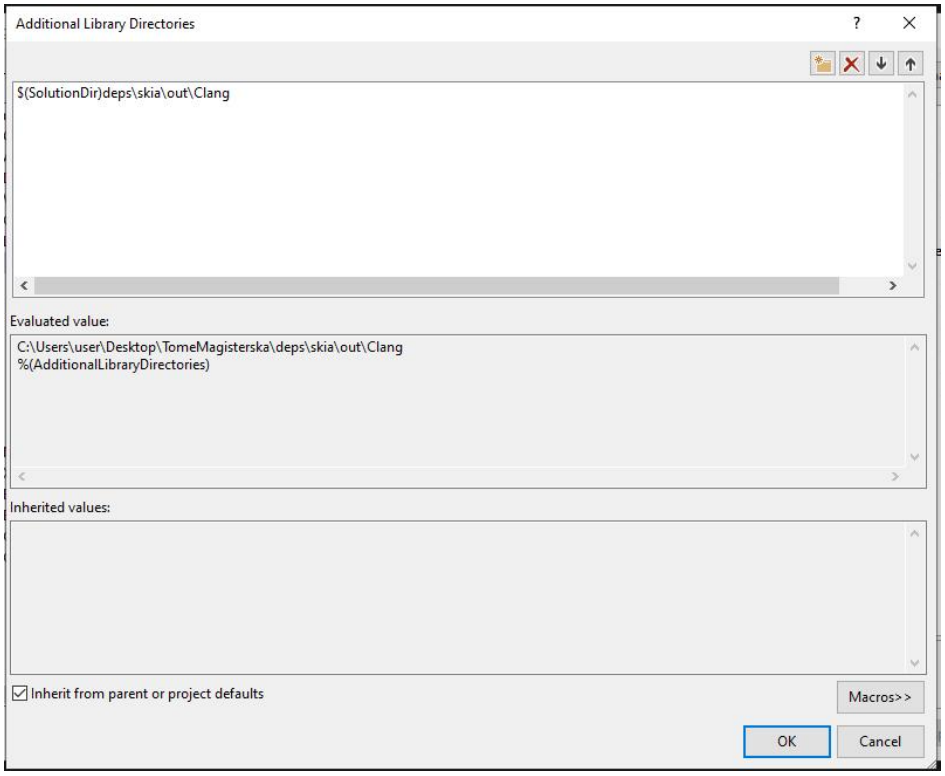
истото име. Другите SK_ константи се од библиотеката Skia и тие се потребни за да се вклучат различни делови на код, како што се различни backends. Во Skia што ја компајлираме, кажано е на компајлерот да ги користи тие backend имплементации. Потребни се константите за да добиеме пристап до специфичните повици дефинирани во хедер фајловите во Skia. SK_GANESH и SK_GL се специфично за backend на ANGLE. Harfbuzz и ICU се библиотеки за текст. Harfbuzz е библиотека за обликување на текст.

3. Во Configuration Properties -> C/C++ -> Code Generation се поставува во Runtime Library вредност од “Multi-threaded (/MT)”.



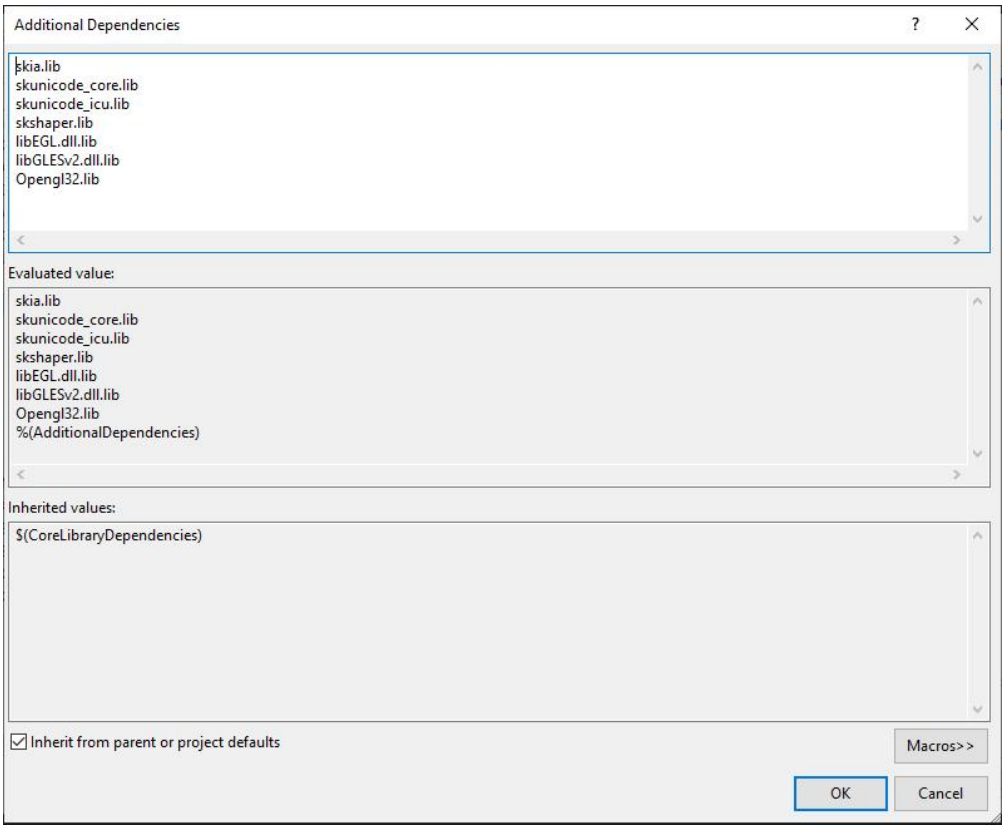

*Слика 26. Поставување на Runtime Library во /MT*

4. Во Configuration Properties -> Linker -> General се поставуваат патеките каде се наоѓаат .lib искомпајлираните објекти во Skia. Во Additional Library Directories, ја поставуваме следната ставка:
   - $(SolutionDir)deps\skia\out\Clang



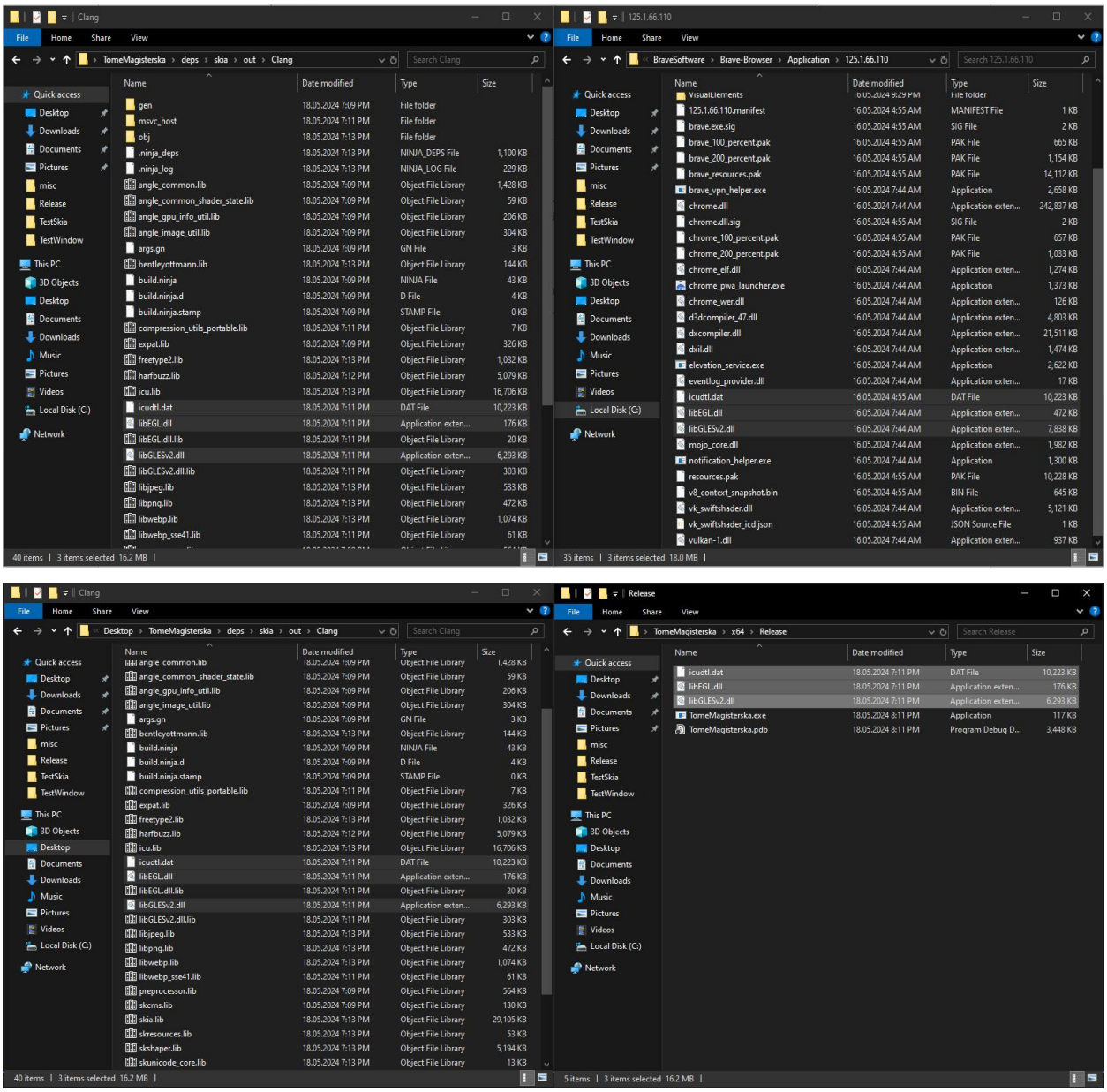

*Слика 27. Поставување на Additional Library Directories*

5. Во Configuration Properties -> Linker -> Input се поставуваат следните ставки во Additional Dependencies:
   - skia.lib
   - skunicode_core.lib
   - skunicode_icu.lib
   - skshaper.lib
   - libEGL.dll.lib
   - libGLESv2.dll.lib
   - Opengl32.lib



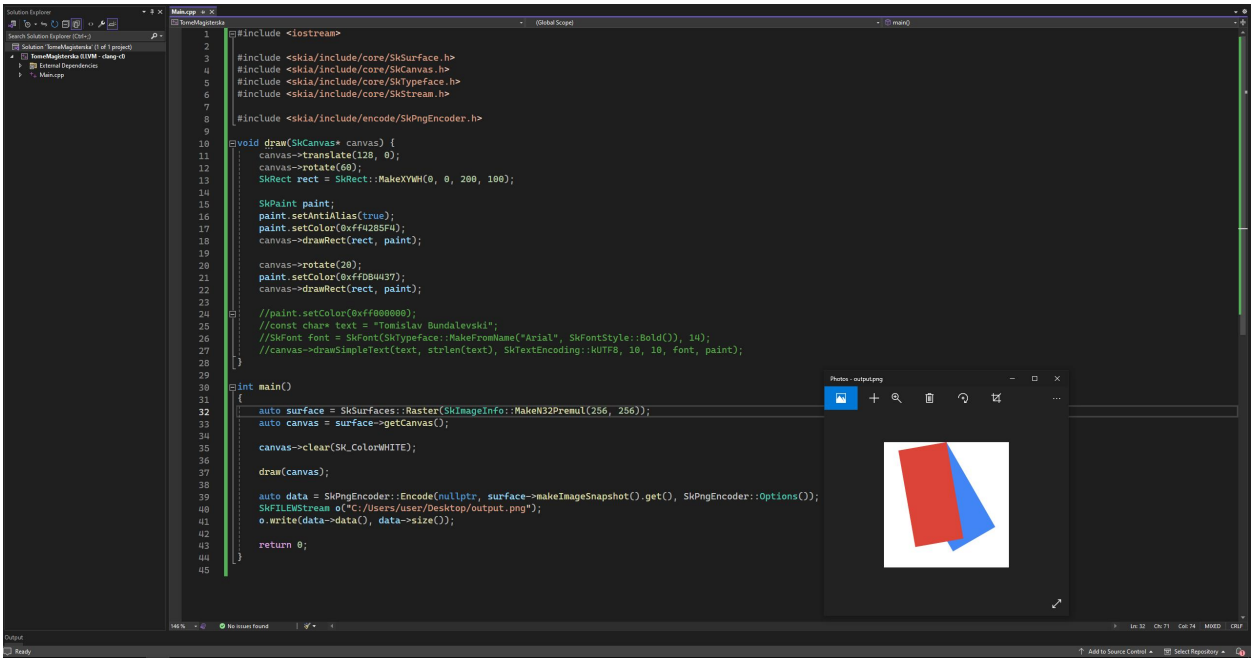

*Слика 28. Поставување на Additional Dependencies*

Ова е последниот чекор за поврзување со Skia. Наредeн чекор е да се ископираат фајловите потребни кога ќе започне програмот, кога се користи Skia во директориумот каде што се наоѓа .exe програмот. Истите фајлови може да се најдат во познатите апликации кои ја користат библиотеката Skia или framework Chromium.




*Слика 29. Копирање на потребните фајлови. Истите фајлови може да се најдат во другите познати софтвери како и прелистувачите*

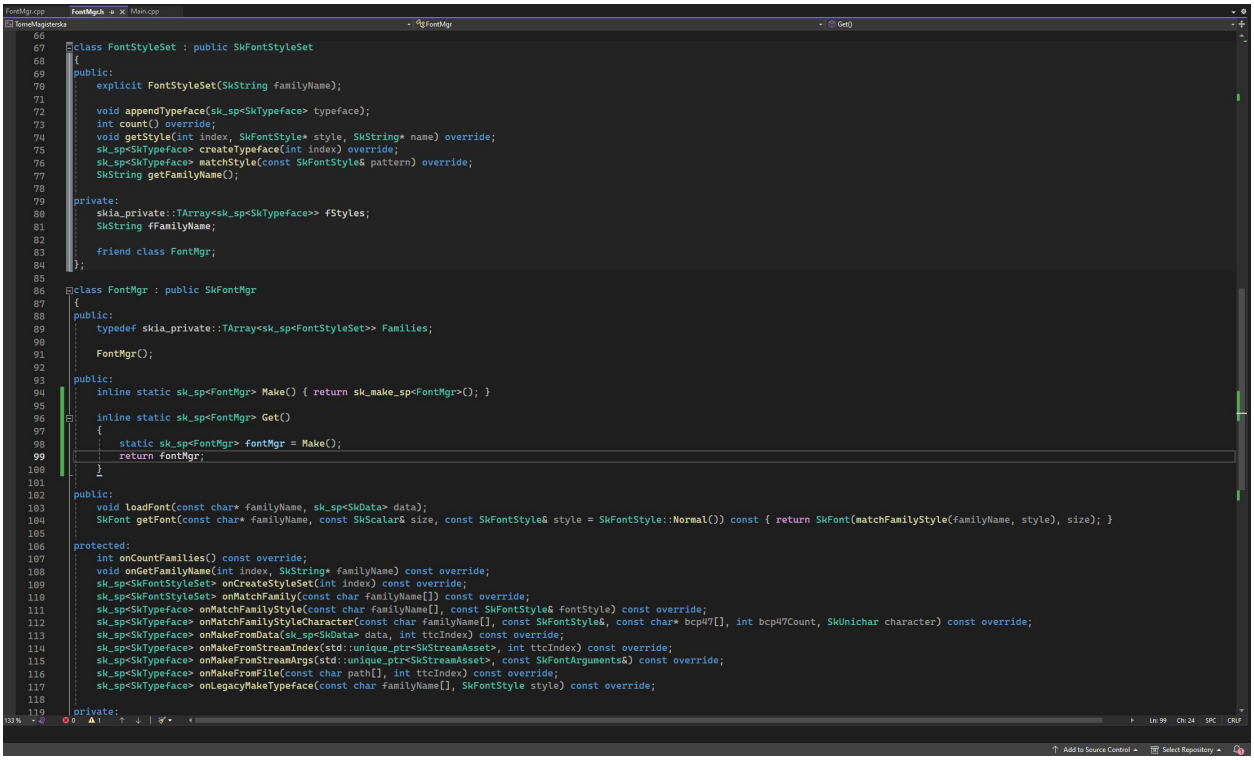
Да се тестира и демонстрира дали е сè поврзано како што треба, ќе се продуцира истата слика од претходно. На тој начин, ќе знаеме дали барем цртањето со CPU работи како што треба.




*Слика 30. Тестирање на Skia*

Тестирањето со CPU цртање помина одлично! Но, ако можеме да забележиме, дел од сликата фали: текстот! Ако се тргна коментарот кај делот со текст и фонт, ќе добиеме компајлерска грешка. Сликата која беше направена и претставена од претходно е направена преку онлајн (online) алатка која интерно користи Skia, но некоја постара верзија. Во проектот, најактуелната верзија на Skia се користи и има многу голема разлика со таа  што е онлајн верзија . За да се овозможи текст, Skia го тера клиентот да ги менаџира своите фонтови, наместо Skia да го прави тоа интерно на свој начин. Skia нуди класа за менаџирање со фонтови и таа се вика SkFontMgr.

Да се направи прилагодена имплементација која на старт на програмот, ќе му побара на оперативниот систем да ги вчита во меморија неколкуте познати фонтови по наш избор.



```
class FontStyleSet : public SkFontStyleSet
{
public:
    explicit FontStyleSet(SkString familyName);

    void appendTypeface(sk_sp<SkTypeface> typeface);
    int count() override;
    void getStyle(int index, SkFontStyle* style, SkString* name) override;
    sk_sp<SkTypeface> createTypeface(int index) override;
    sk_sp<SkTypeface> matchStyle(const SkFontStyle& pattern) override;
    SkString getFamilyName();

private:
    skia_private::TArray<sk_sp<SkTypeface>> fStyles;
    SkString fFamilyName;

    friend class FontMgr;
};

class FontMgr : public SkFontMgr
{
public:
    typedef skia_private::TArray<sk_sp<FontStyleSet>> Families;

    FontMgr();

public:
    inline static sk_sp<FontMgr> Make() { return sk_make_sp<FontMgr>(); }

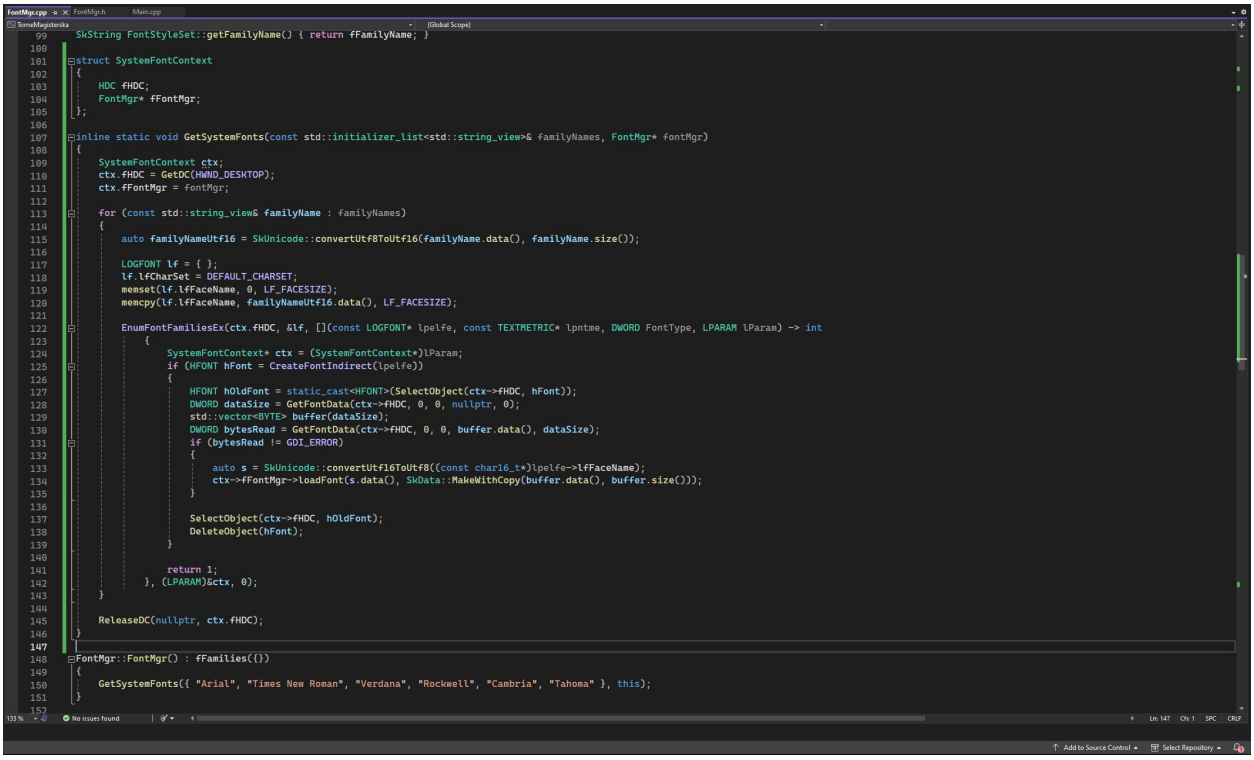
    inline static sk_sp<FontMgr> Get()
    {
        static sk_sp<FontMgr> fontMgr = Make();
        return fontMgr;
    }

public:
    void loadFont(const char* familyName, sk_sp<SkData> data);
    SkFont getFont(const char* familyName, const SkScalar& size, const SkFontStyle& style = SkFontStyle::Normal()) const { return SkFont(matchFamilyStyle(familyName, style), size); }

protected:
    int onCountFamilies() const override;
    void onGetFamilyName(int index, SkString* familyName) const override;
    sk_sp<SkFontStyleSet> onCreateStyleSet(int index) const override;
    sk_sp<SkFontStyleSet> onMatchFamily(const char familyName[]) const override;
    sk_sp<SkTypeface> onMatchFamilyStyle(const char familyName[], const SkFontStyle& fontStyle) const override;
    sk_sp<SkTypeface> onMatchFamilyStyleCharacter(const char familyName[], const SkFontStyle&, const char* bcp47[], int bcp47Count, SkUnichar character) const override;
    sk_sp<SkTypeface> onMakeFromData(sk_sp<SkData> data, int ttcIndex) const override;
    sk_sp<SkTypeface> onMakeFromStreamIndex(std::unique_ptr<SkStreamAsset>, int ttcIndex) const override;
    sk_sp<SkTypeface> onMakeFromStreamArgs(std::unique_ptr<SkStreamAsset>, const SkFontArguments&) const override;
    sk_sp<SkTypeface> onMakeFromFile(const char path[], int ttcIndex) const override;
    sk_sp<SkTypeface> onLegacyMakeTypeface(const char familyName[], SkFontStyle style) const override;

private:
```

*Слика 31. Делумна декларација на фонт менаџер "FontMgr"*



```
SkString FontStyleSet::getFamilyName() { return fFamilyName; }

struct SystemFontContext
{
    HDC fHDC;
    FontMgr* fFontMgr;
};

inline static void GetSystemFonts(const std::initializer_list<std::string_view>& familyNames, FontMgr* fontMgr)
{
    SystemFontContext ctx;
    ctx.fHDC = GetDC(HWND_DESKTOP);
    ctx.fFontMgr = fontMgr;

    for (const std::string_view& familyName : familyNames)
    {
        auto familyNameUtf16 = SkUnicode::convertUtf8ToUtf16(familyName.data(), familyName.size());

        LOGFONT lf = { };
        lf.lfCharSet = DEFAULT_CHARSET;
        memset(lf.lfFaceName, 0, LF_FACESIZE);
        memcpy(lf.lfFaceName, familyNameUtf16.data(), LF_FACESIZE);

        EnumFontFamiliesEx(ctx.fHDC, &lf, [](const LOGFONT* lpelfe, const TEXTMETRIC* lpntme, DWORD FontType, LPARAM lParam) -> int
            {
                SystemFontContext* ctx = (SystemFontContext*)lParam;
                if (HFONT hFont = CreateFontIndirect(lpelfe))
                {
                    HFONT hOldFont = static_cast<HFONT>(SelectObject(ctx->fHDC, hFont));
                    DWORD dataSize = GetFontData(ctx->fHDC, 0, 0, nullptr, 0);
                    std::vector<BYTE> buffer(dataSize);
                    DWORD bytesRead = GetFontData(ctx->fHDC, 0, 0, buffer.data(), dataSize);
                    if (bytesRead != GDI_ERROR)
                    {
                        auto s = SkUnicode::convertUtf16ToUtf8((const char16_t*)lpelfe->lfFaceName);
                        ctx->fFontMgr->loadFont(s.data(), SkData::MakeWithCopy(buffer.data(), buffer.size()));
                    }

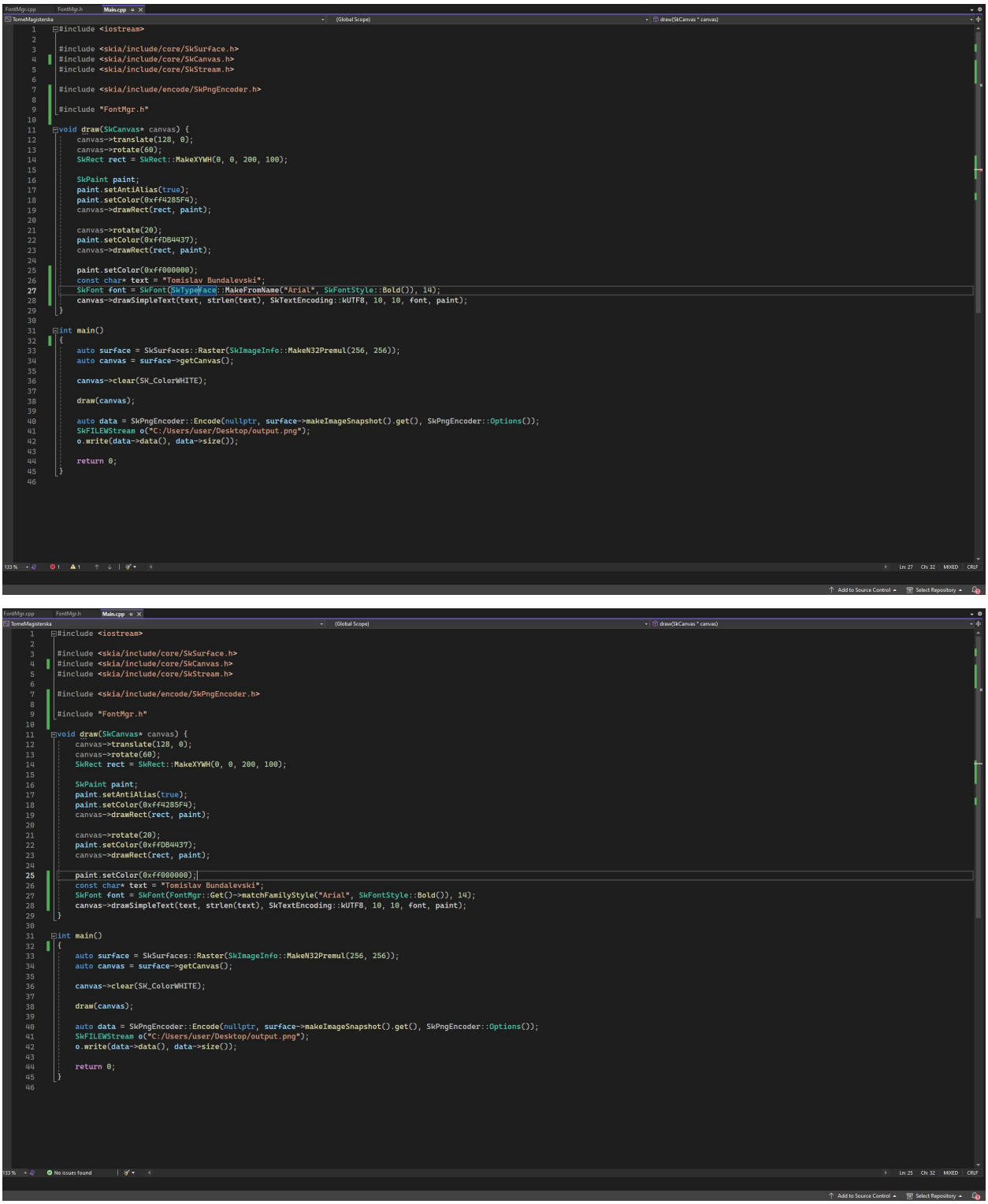
                    SelectObject(ctx->fHDC, hOldFont);
                    DeleteObject(hFont);
                }

                return 1;
            }, (LPARAM)&ctx, 0);
    }

    ReleaseDC(nullptr, ctx.fHDC);
}
FontMgr::FontMgr() : fFamilies({})
{
    GetSystemFonts({ "Arial", "Times New Roman", "Verdana", "Rockwell", "Cambria", "Tahoma" }, this);
}
```

*Слика 32. Делумна имплементација на фонт менаџер "FontMgr"*

Следно, може прилагодениот фонт менаџер да се искористи за да може да се креира коректен фонт.



```cpp
#include <iostream>

#include <skia/include/core/SkSurface.h>
#include <skia/include/core/SkCanvas.h>
#include <skia/include/core/SkStream.h>

#include <skia/include/encode/SkPngEncoder.h>

#include "FontMgr.h"

void draw(SkCanvas* canvas) {
    canvas->translate(128, 0);
    canvas->rotate(60);
    SkRect rect = SkRect::MakeXYWH(0, 0, 200, 100);

    SkPaint paint;
    paint.setAntiAlias(true);
    paint.setColor(0xff4285F4);
    canvas->drawRect(rect, paint);

    canvas->rotate(20);
    paint.setColor(0xffDB4437);
    canvas->drawRect(rect, paint);

    paint.setColor(0xff000000);
    const char* text = "Tomislav Bundalevski";
    SkFont font = SkFont(SkTypeface::MakeFromName("Arial", SkFontStyle::Bold()), 14);
    canvas->drawSimpleText(text, strlen(text), SkTextEncoding::kUTF8, 10, 10, font, paint);
}

int main()
{
    auto surface = SkSurfaces::Raster(SkImageInfo::MakeN32Premul(256, 256));
    auto canvas = surface->getCanvas();

    canvas->clear(SK_ColorWHITE);

    draw(canvas);

    auto data = SkPngEncoder::Encode(nullptr, surface->makeImageSnapshot().get(), SkPngEncoder::Options());
    SkFILEWStream o("C:/Users/user/Desktop/output.png");
    o.write(data->data(), data->size());

    return 0;
}
```



```cpp
#include <iostream>

#include <skia/include/core/SkSurface.h>
#include <skia/include/core/SkCanvas.h>
#include <skia/include/core/SkStream.h>

#include <skia/include/encode/SkPngEncoder.h>

#include "FontMgr.h"

void draw(SkCanvas* canvas) {
    canvas->translate(128, 0);
    canvas->rotate(60);
    SkRect rect = SkRect::MakeXYWH(0, 0, 200, 100);

    SkPaint paint;
    paint.setAntiAlias(true);
    paint.setColor(0xff4285F4);
    canvas->drawRect(rect, paint);

    canvas->rotate(20);
    paint.setColor(0xffDB4437);
    canvas->drawRect(rect, paint);

    paint.setColor(0xff000000);
    const char* text = "Tomislav Bundalevski";
    SkFont font = SkFont(FontMgr::Get()->matchFamilyStyle("Arial", SkFontStyle::Bold()), 14);
    canvas->drawSimpleText(text, strlen(text), SkTextEncoding::kUTF8, 10, 10, font, paint);
}

int main()
{
    auto surface = SkSurfaces::Raster(SkImageInfo::MakeN32Premul(256, 256));
    auto canvas = surface->getCanvas();

    canvas->clear(SK_ColorWHITE);

    draw(canvas);

    auto data = SkPngEncoder::Encode(nullptr, surface->makeImageSnapshot().get(), SkPngEncoder::Options());
    SkFILEWStream o("C:/Users/user/Desktop/output.png");
    o.write(data->data(), data->size());

    return 0;
}
```

*Слика 33. Препрaвање во точна синтакса кодот за вчитување на фонт и цртање на текст*

Нормално, кога ќе се изврши кодот, се добива една многу слична и “фамилијарна”, односно позната слика:

```
void draw(SkCanvas* canvas) {
    canvas->translate(128, 0);
    canvas->rotate(60);
    SkRect rect = SkRect::MakeXYWH(0, 0, 200, 100);

    SkPaint paint;
    paint.setAntiAlias(true);
    paint.setColor(0xff4285F4);
    canvas->drawRect(rect, paint);

    canvas->rotate(20);
    paint.setColor(0xffDB4437);
    canvas->drawRect(rect, paint);

    paint.setColor(0xff000000);
    const char* text = "Tomislav Bundalevski";
    SkFont font = SkFont(FontMgr::Get()->matchFamilyStyle("Arial", SkFontStyle::Bold()), 14);
    canvas->drawSimpleText(text, strlen(text), SkTextEncoding::kUTF8, 10, 10, font, paint);
}

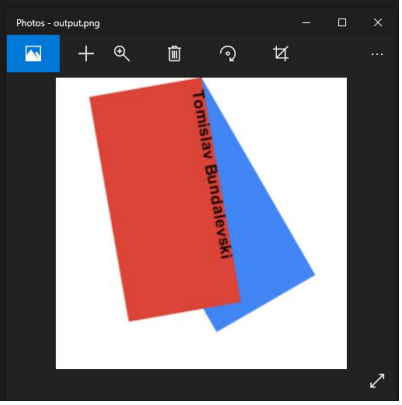
int main()
{
    auto surface = SkSurfaces::Raster(SkImageInfo::MakeN32Premul(256, 256));
    auto canvas = surface->getCanvas();

    canvas->clear(SK_ColorWHITE);

    draw(canvas);

    auto data = SkPngEncoder::Encode(nullptr, surface->makeImageSnapshot().get(), SkPngEncoder::Options());
    SkFILEWStream o("C:/Users/user/Desktop/output.png");
    o.write(data->data(), data->size());

    return 0;
}
```




*Слика 34. Тестирање на Skia*

Skia е најдобро користен од софтвери имплементирани од Google, поради тоа што и Skia е направен од Google. Chromium прелистувачите имаат можност да му кажат на корисникот кои Skia команди се направени за даден вчитан HTML документ. Знаејќи го тоа, и исто така, знаејќи дека можностите се широко отворени за имплементација на софтвер како UI framework или алатка за визуелно програмирање, може да се започне со понатамошни чекори.

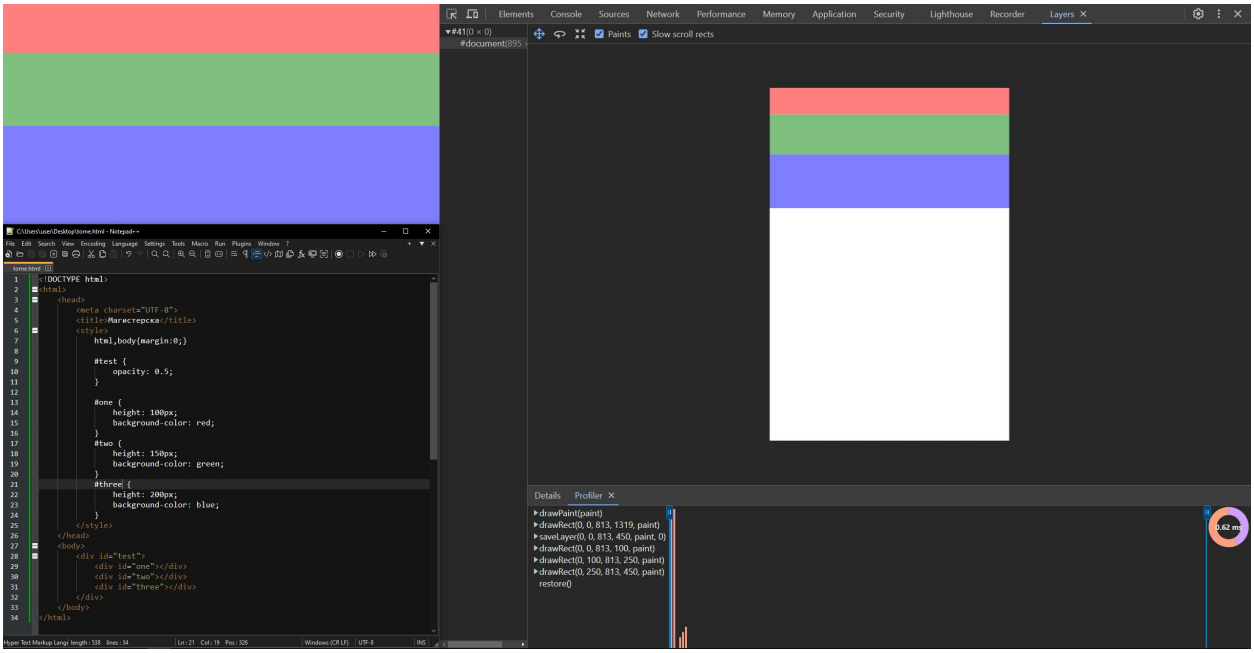
На пример, при набљудување и анализа на тие команди, можеме да стигнеме до свои имплементации на CSS својства (пр. *box-shadow*, *outline*, *border*, *border-radius*, *filter*, *transform*, *opacity*, *background-color*, *background-image*, *overflow*, *text-align*, итн.). При промена на тие CSS својства и повторно читање на HTML документот, возможно е да се стигне до интерната логика на Chromium кога, како и со кои параметри ги извршува тие команди за да стигне до посебен резултат на екран. Со тоа, возможно е да се направат „градежните блокови” на едно GUI дрво (пр. целосна/делумна имплементација на CSS box модел како во прелистувачите, имплементација на текст CSS својства како *text-align*,

*letter-spacing*, *word-spacing*, итн., слики и слично). Можно е да се отиде и чекор понапред од тоа и да се имплементира во истото GUI дрво и автоматски layout на елементите (и со тоа и дополнителни имплементации на CSS, итн.).

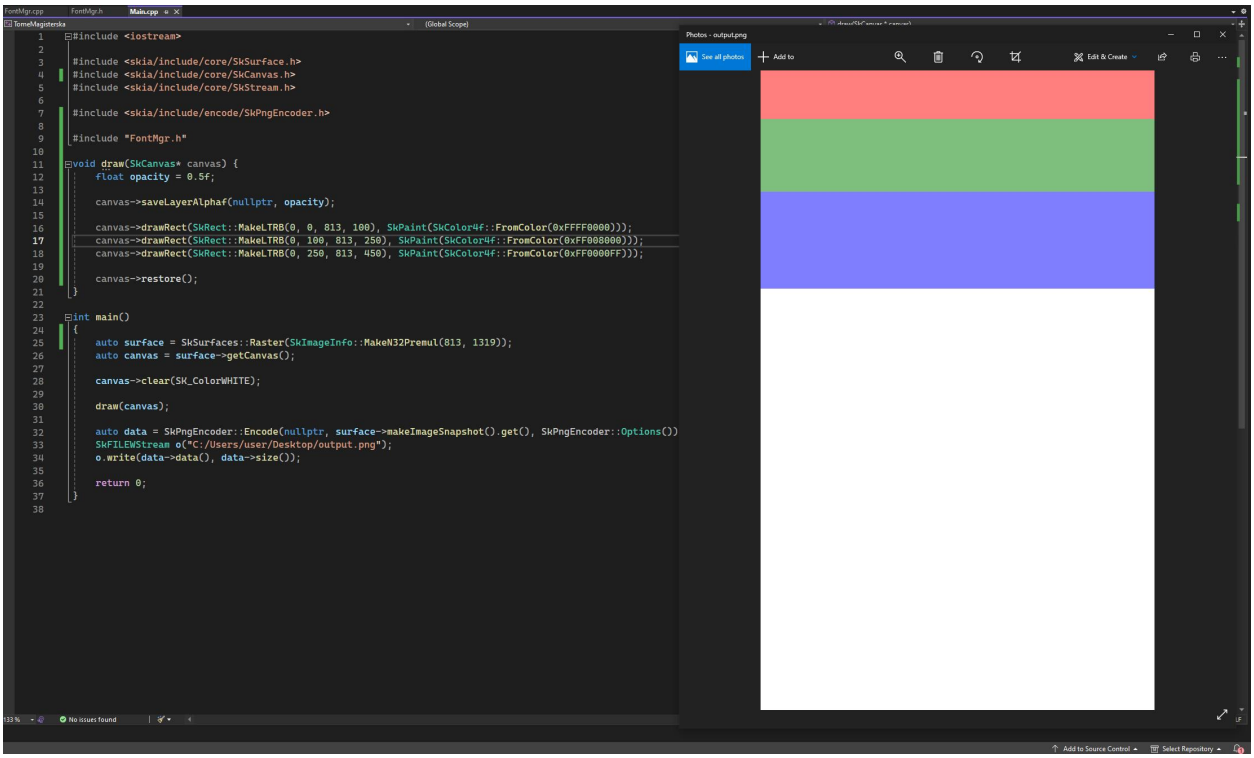
Тоа е само еден од многуте правци во однос на тоа како и што би можеле да направиме со библиотеката.

Друг правец е да се креира посебна алатка како во Unreal Engine за визуелно програмирање, преку јазли. Тие јазли би се цртале на еден начин генерално, но инпутите и аутпутите треба да бидат динамични. Пиновите од аутпутите на еден јазол кои би се конектирале со инпутите на други јазли исто така би требало да можат да се влечат и истите динамично да се цртаат врз основа каде се влече тој пин и каде му е неговата првобитна локација.

Следно, ќе се покаже пример како може едно CSS својство, како opacity, да се имплементира, следејќи ги инструкциите на Chromium прелистувачот.

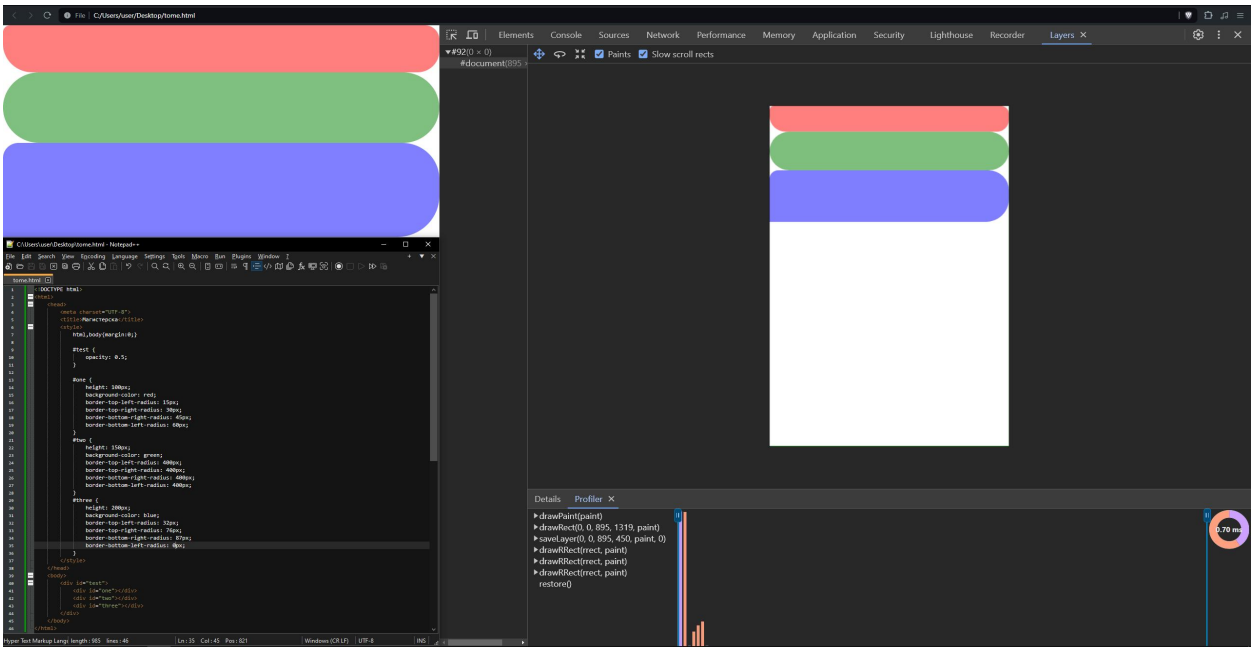



*Слика 35. Chromium Skia команди генерирани од едно HTML дрво со CSS својства (правоаголници)*

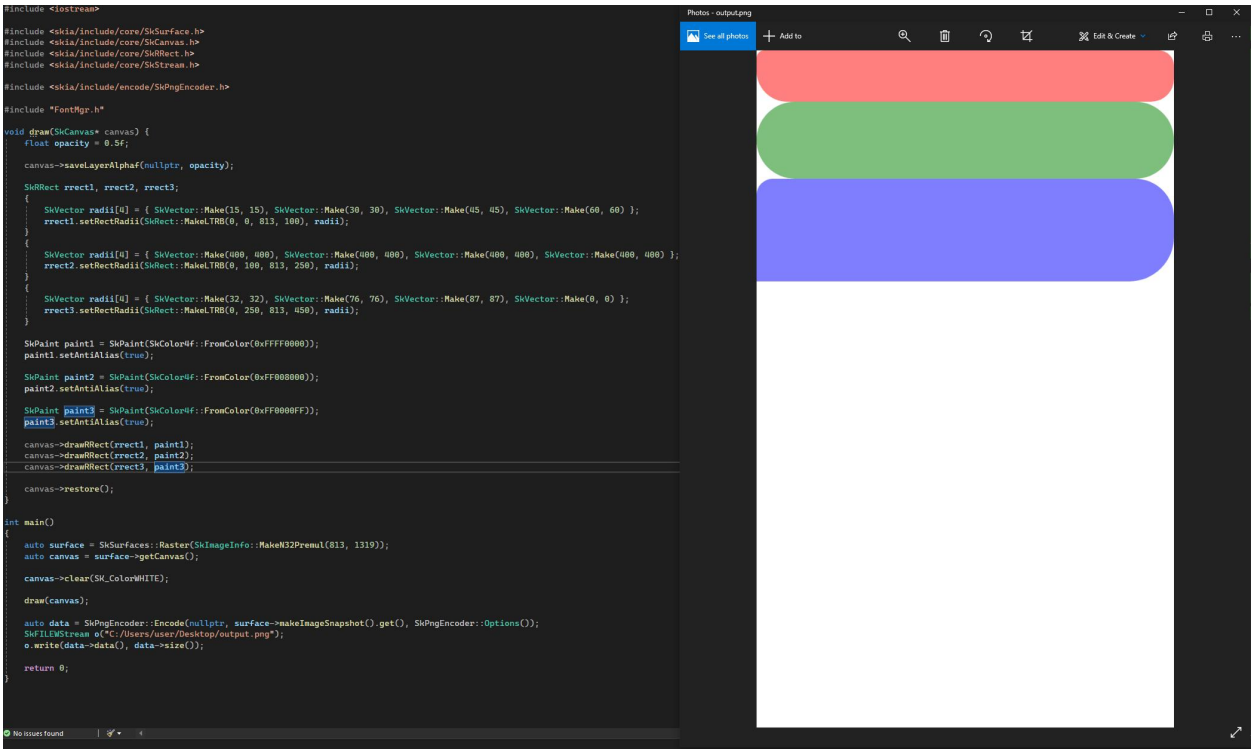



*Слика 36. Истите Skia команди извршени од проектот (правоаголници)*

Како што може да се забележи, нема ниту мала разлика во аутпут од Chromium и од кодот погоре. Следно, ќе биде прикажан обид да се нацртаат заоблени правоаголници, односно истото од горе, но преку употреба на border-radius, да се добие ист цртеж како кај Chromium.

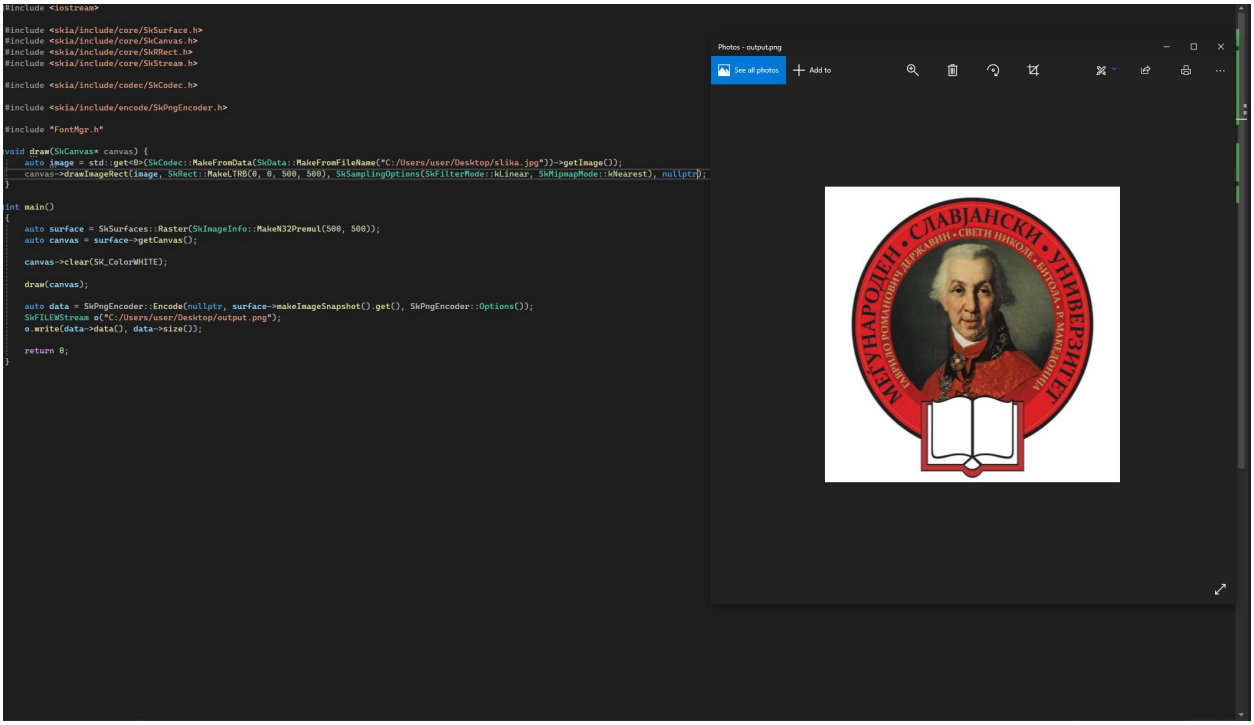


*Слика 37. Chromium Skia команди генерирани од едно HTML дрво со CSS својства (заоблени правоаголници)*



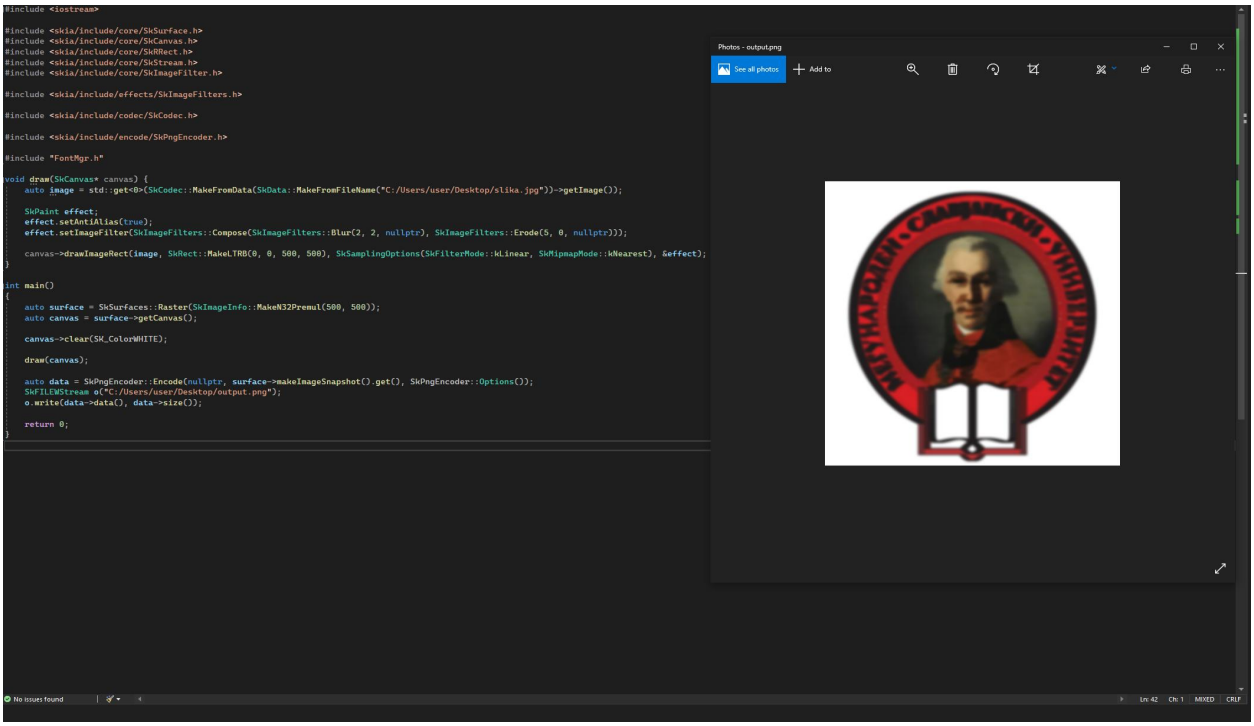

*Слика 38. Истите Skia команди извршени од проектот (заоблени правоаголници)*

Повторно се добива ист аутпут како и прелистувачот. Градежните блокови за цртање на прелистувачот се прикажани и демонстрирани (правоаголници, заоблени правоаголници, текст, итн.). Една слика е исто така многу лесно да се вчита и да се нацрта на екран. Следно ќе се демонстрира и тоа.




*Слика 39. Цртање на слика преку Skia*

Како и во познати софтвери како Photoshop, постојат и опции како и ефекти да се нанесат на секој “draw” повик на canvas. Следно, ќе се покаже еден пример како се користат филтри за ефекти на слика да се замагли и еродира:




*Слика 40. Цртање на слика со ефекти преку Skia*

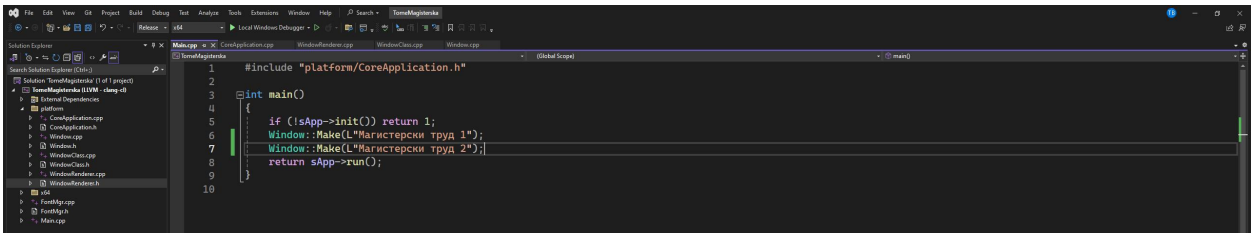
Со тоа, се демонстрира Skia колку е моќна и колку работи поддржува. Може да се сумира што е направено до сега:

1. Се искомпајлира Skia и статички ја поврзавме со нашиот проект.
2. Тоа дозволува да се црта каде било без повикување до оперативниот систем, освен за фонтови.
3. Со Skia успешно и лесно се цртаат примитивни елементи како правоаголници, заоблени правоаголници, текст, слики и ефекти.
4. Досега, се користеше растерска површина, односно canvas, чиишто команди за цртање, сечење, итн. се извршуваат од страна на CPU. Иако Skia е искомпајлирана со GPU backend, начинот на кој се креира "GPU canvas" многу варира од тој преку CPU, односно Raster.
5. Без разлика дали цртањето се врши преку CPU или GPU, нема разлика во аутпут на екранот од другите големи софтвери како Chromium прелистувачот.

Следно, ќе биде демонстриран начин на кој може еден UI framework да се постави за да се овозможи креирање на Windows прозорци и на истите прозорци да се црта преку GPU. Понатаму, ќе се истражат покомплексни CSS својства како border, outline и box-shadow, напредни текст операции (пр. обликување на текст, селектирање на текст, CSS својства на текст како *text-align*, *letter-spacing*, *word-spacing*, итн.) и автоматски layout преку користење на библиотека, но модифицирана за конкретни потреби. Со тоа, му се овозможува на крајниот корисник на т.н. framework да може да изгради свое дрво (пр. преку HTML) и да ги предаде својствата (пр. преку CSS) на индивидуалните јазли, односно елементи и да си направи свој framework или апликација користејќи го постоечкото што го нуди т.н. framework.

Во проектот, ќе биде дефиниран еден фолдер "platform" каде што ќе се дефинира Windows Message Loop за настани кои ќе се случуваат на самата апликација, односно во самите прозорци. На почеток на апликацијата, апликацијата треба да ги регистрира прозорците како класи кои понатаму ќе ги користи. Потоа, треба да се креира барем еден прозорец за да самата апликација почне да работи. Откога ќе се креира самиот прозорец, треба да креираме еден SkSurface/SkCanvas за цртање преку GPU кои што е „закачен" на тој прозорец. Тоа ќе се направи со помош на ANGLE. Самиот прозорец треба да одговара

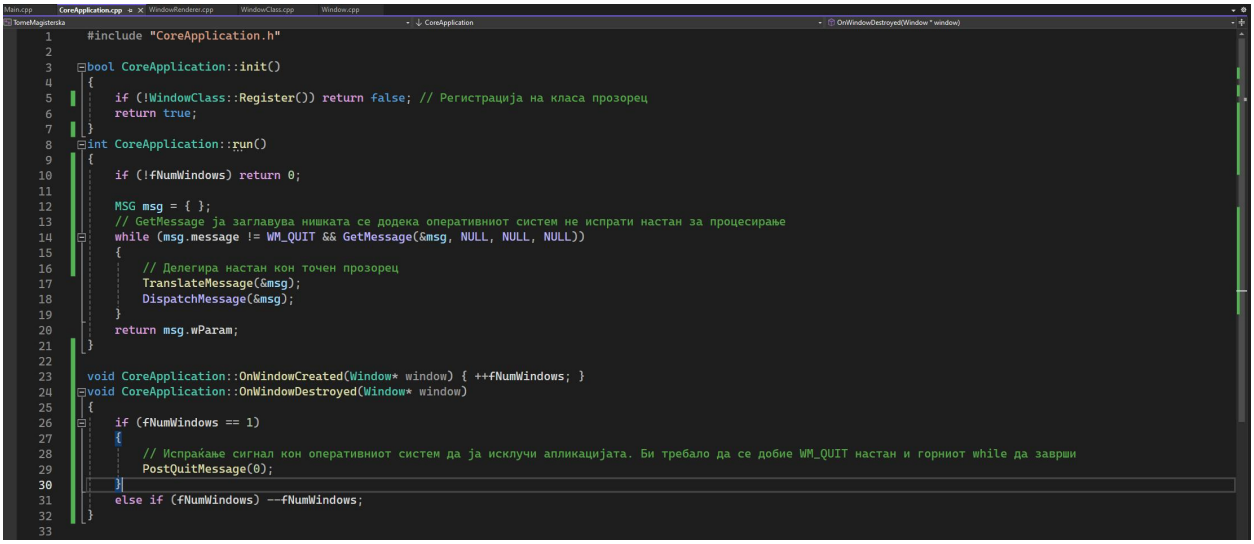
на настани кои оперативниот систем му ги доделува на него преку праќање порака, односно сигнал, односно настан преку Windows Message Loop и тој му го делегира настанот на точниот прозорец кој треба да го прими тој настан и да го процесира соодветно. Кога последниот прозорец ќе се затвори, апликацијата испраќа сигнал до оперативниот систем дека треба да ја исклучи апликацијата.



```
#include "platform/CoreApplication.h"

int main()
{
    if (!sApp->init()) return 1;
    Window::Make(L"Магистерски труд 1");
    Window::Make(L"Магистерски труд 2");
    return sApp->run();
}
```

*Слика 41. Креирање на апликација со 2 прозорци*



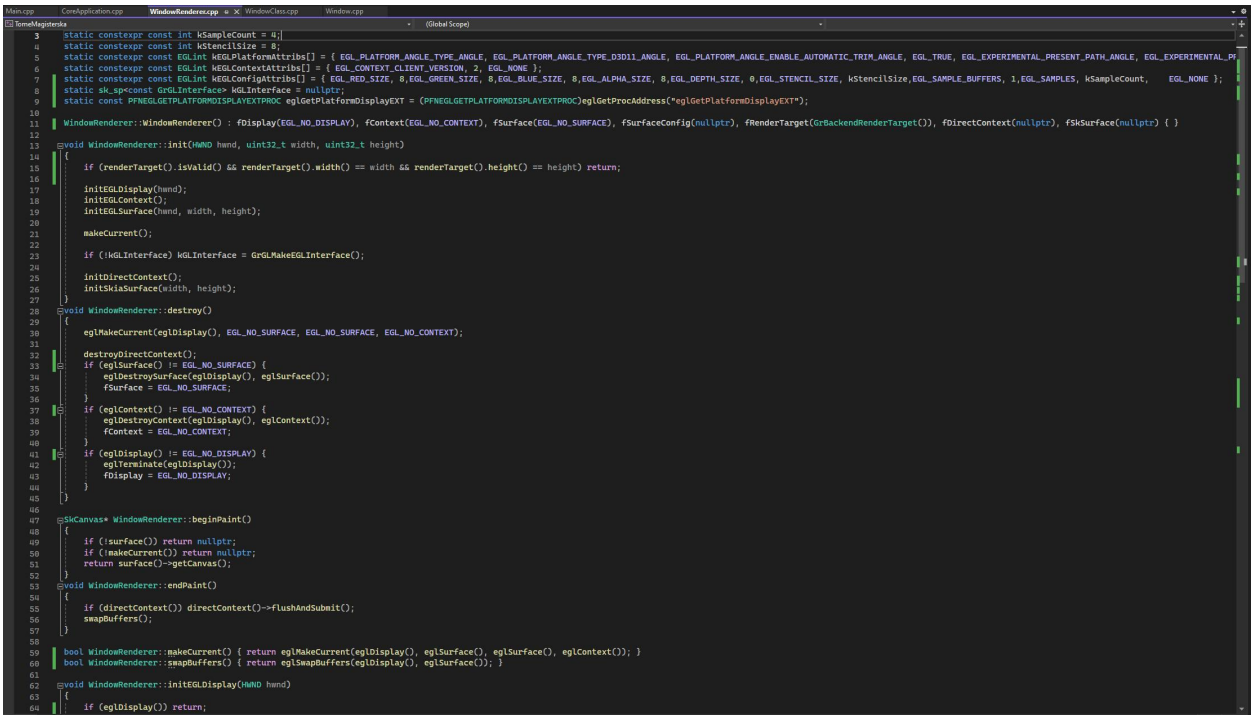
```
#include "CoreApplication.h"

bool CoreApplication::init()
{
    if (!WindowClass::Register()) return false; // Регистрација на класа прозорец
    return true;
}
int CoreApplication::run()
{
    if (!fNumWindows) return 0;

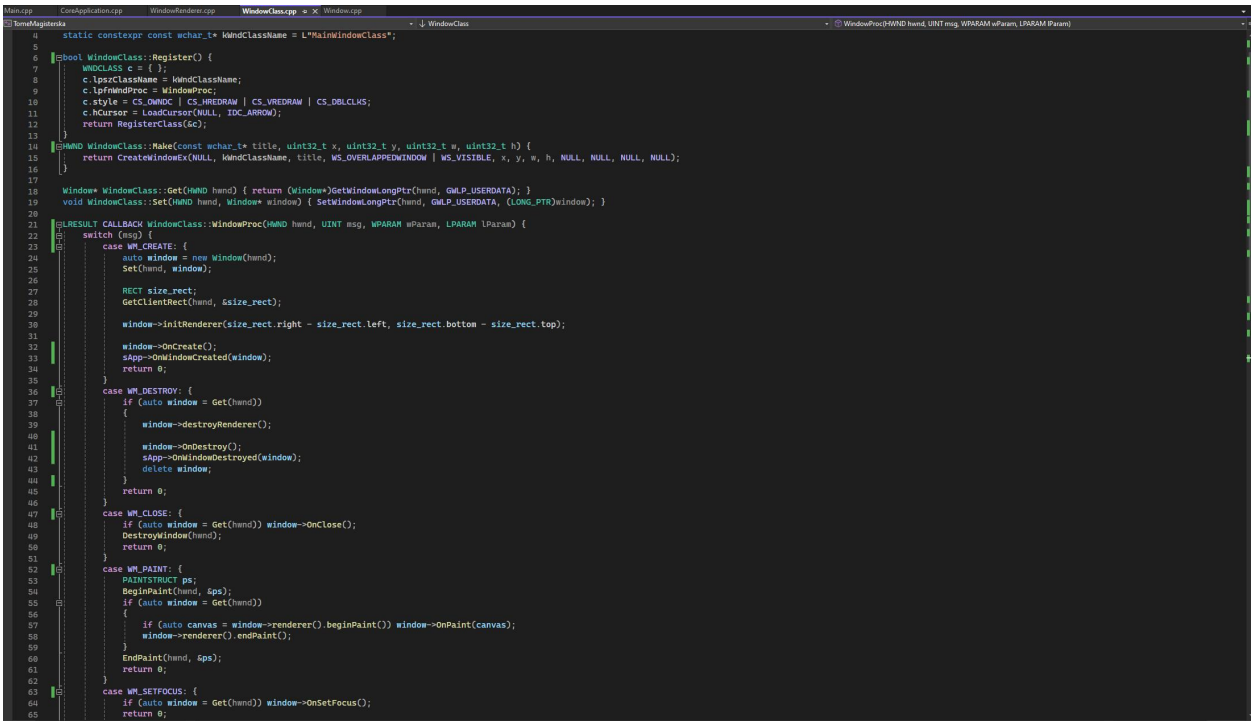
    MSG msg = { };
    // GetMessage ја заглавува нишката се додека оперативниот систем не испрати настан за процесирање
    while (msg.message != WM_QUIT && GetMessage(&msg, NULL, NULL, NULL))
    {
        // Делегира настан кон точен прозорец
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return msg.wParam;
}

void CoreApplication::OnWindowCreated(Window* window) { ++fNumWindows; }
void CoreApplication::OnWindowDestroyed(Window* window)
{
    if (fNumWindows == 1)
    {
        // Испраќање сигнал кон оперативниот систем да ја исклучи апликацијата. Би требало да се добие WM_QUIT настан и горниот while да заврши
        PostQuitMessage(0);
    }
    else if (fNumWindows) --fNumWindows;
}
```

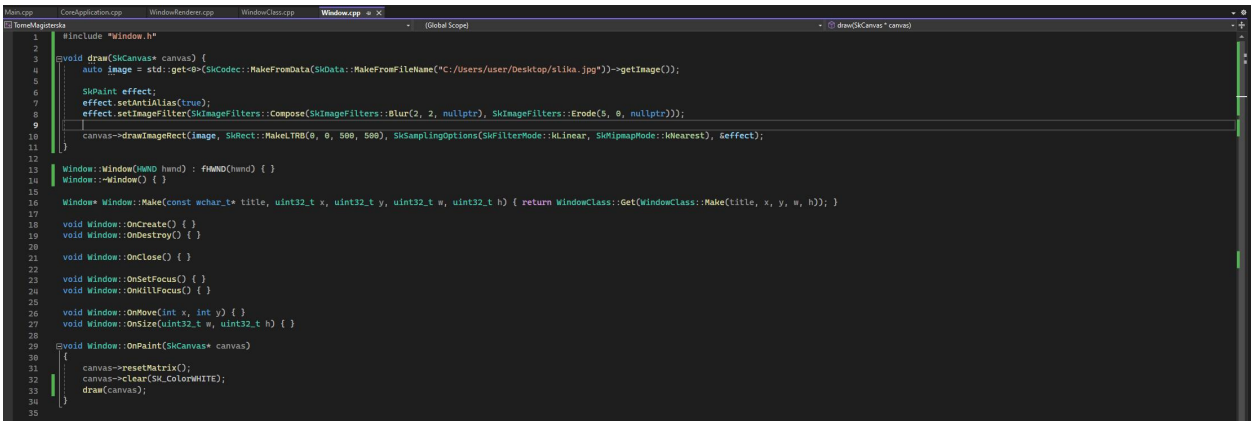
*Слика 42. Windows Message Loop*



*Слика 43. Иницијализација на ANGLE со Skia*



*Слика 44. Регистрирање на класа, креирање на класа и справување со настани за еден прозорец*



```cpp
#include "Window.h"

void draw(SkCanvas* canvas) {
    auto image = std::get<0>(SkCodec::MakeFromData(SkData::MakeFromFileName("C:/Users/user/Desktop/slika.jpg"))->getImage());

    SkPaint effect;
    effect.setAntiAlias(true);
    effect.setImageFilter(SkImageFilters::Compose(SkImageFilters::Blur(2, 2, nullptr), SkImageFilters::Erode(5, 0, nullptr)));

    canvas->drawImageRect(image, SkRect::MakeLTRB(0, 0, 500, 500), SkSamplingOptions(SkFilterMode::kLinear, SkMipmapMode::kNearest), &effect);
}

Window::Window(HWND hwnd) : fHWND(hwnd) { }
Window::~Window() { }

Window* Window::Make(const wchar_t* title, uint32_t x, uint32_t y, uint32_t w, uint32_t h) { return WindowClass::Get(WindowClass::Make(title, x, y, w, h)); }

void Window::OnCreate() { }
void Window::OnDestroy() { }

void Window::OnClose() { }

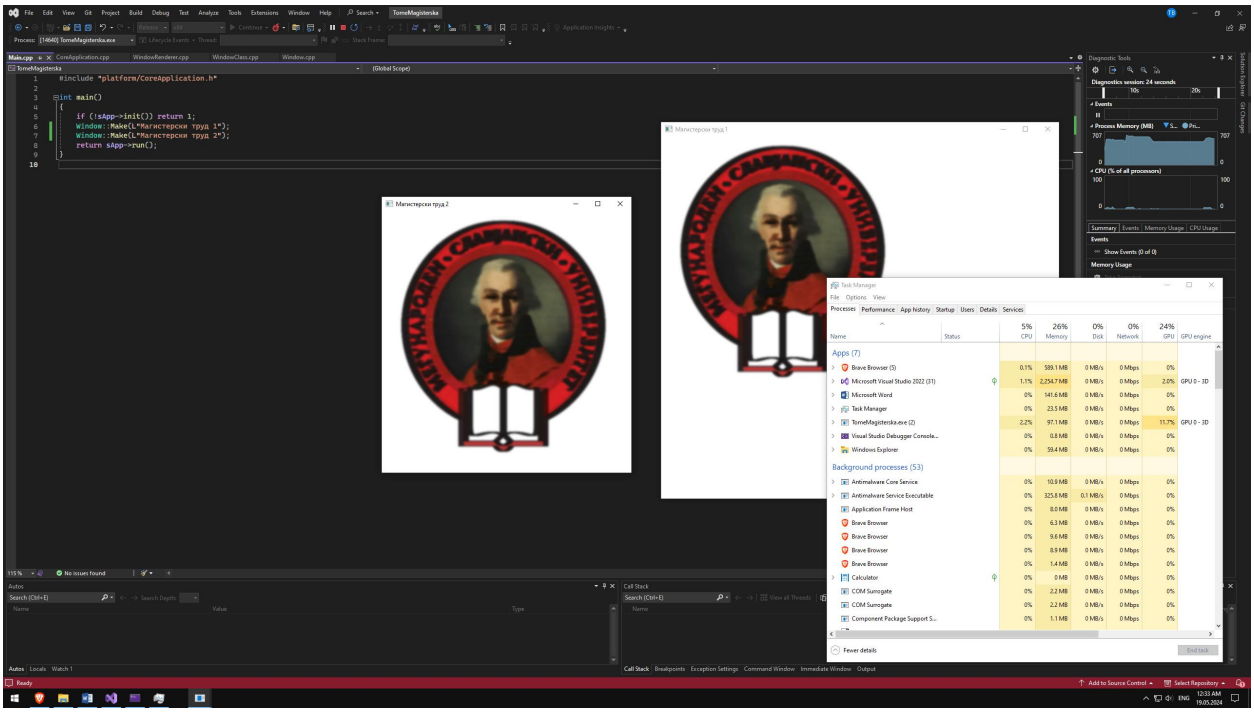
void Window::OnSetFocus() { }
void Window::OnKillFocus() { }

void Window::OnMove(int x, int y) { }
void Window::OnSize(uint32_t w, uint32_t h) { }

void Window::OnPaint(SkCanvas* canvas)
{
    canvas->resetMatrix();
    canvas->clear(SK_ColorWHITE);
    draw(canvas);
}
```

*Слика 45. Имплементација на прозорец*

Резултатот на овој код се 2 прозорци кои што рендерираат со графичката карта (GPU) користејќи ANGLE со Skia:

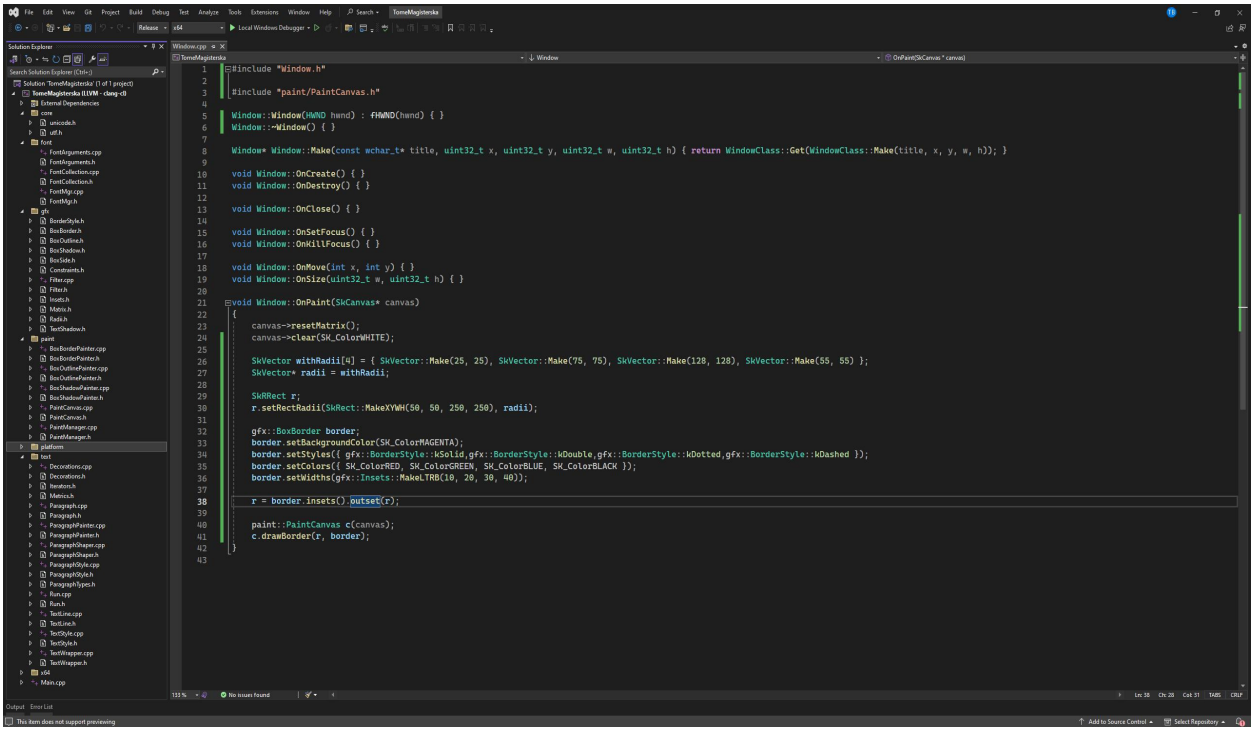


*Слика 46. Прозорци кои што рендерираат со графичката карта користејќи ANGLE со Skia*

Следно, ќе се демонстрираат покомплексни, но често употребени CSS својства кога се изработува една веб- страница, како што се border, outline и box-shadow.

Во изворниот код ќе се дефинираат многу повеќе папки, во кои што има изворен код кој бил анализиран преку користење на прелистувачот со многу различни параметри, со цел да се мапираат истите инструкции како што прелистувачот Chromium ги изведува. Не само тоа, туку и самата геометрија за каде да се црта, со кои параметри, со кои инструкции, со кои исечоци на екранот, итн. Самиот процес инволвира менување на

параметри и според некоја логика, прелистувачот исто така ја менува својата логика, доведувајќи до момент каде што може самите if услови да се мапираат кои се и кога се случуваат под кои услови. Се вчитува повторно документот се додека сите логични патишта не се откријат. Откога се откриле сите патишта, може да се заклучи дека таа логика, со тие параметри е готова. Овој процес се повторува за сите случаи и сите параметри, коишто играат улога во тоа CSS својство. Ова е потребно и задолжително ако сакаме да постигнеме исто цртање како прелистувачот, но исто така, има логични патишта кои се многу побрзи и за пресметување и за цртање (пр. border: 10px solid black; треба да нацрта **еден унифициран** border околу оригиналната геометрија, којшто е црн, со солиден стил и 10 пиксели дебел, додека очигледно, има и можност да се дефинираат сите 4 borders, со 4 различни бои, дебелини и стилови, кои бараат **минимум 4 различни "draw" повици**).



```cpp
#include "Window.h"

#include "paint/PaintCanvas.h"

Window::Window(HWND hwnd) : fHWND(hwnd) { }
Window::~Window() { }

Window* Window::Make(const wchar_t* title, uint32_t x, uint32_t y, uint32_t w, uint32_t h) { return WindowClass::Get(WindowClass::Make(title, x, y, w, h)); }

void Window::OnCreate() { }
void Window::OnDestroy() { }

void Window::OnClose() { }

void Window::OnSetFocus() { }
void Window::OnKillFocus() { }

void Window::OnMove(int x, int y) { }
void Window::OnSize(uint32_t w, uint32_t h) { }

void Window::OnPaint(SkCanvas* canvas)
{
    canvas->resetMatrix();
    canvas->clear(SK_ColorWHITE);

    SkVector withRadii[4] = { SkVector::Make(25, 25), SkVector::Make(75, 75), SkVector::Make(128, 128), SkVector::Make(55, 55) };
    SkVector* radii = withRadii;

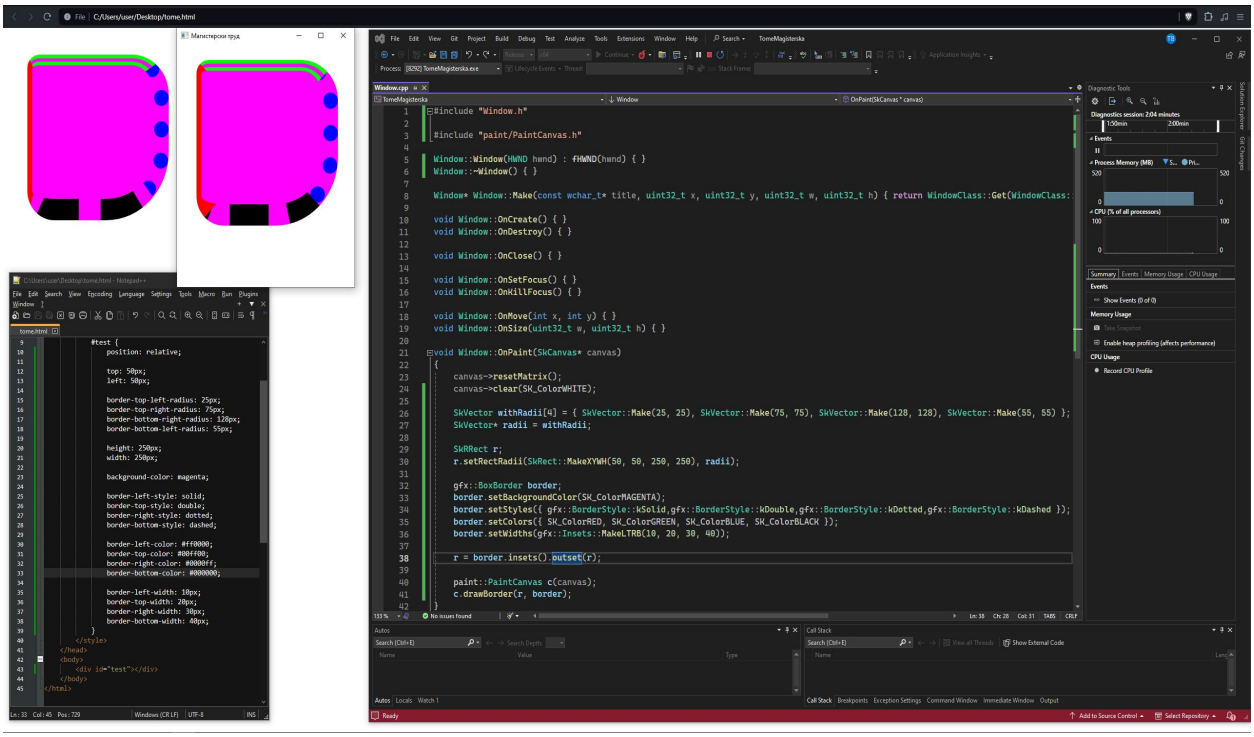
    SkRRect r;
    r.setRectRadii(SkRect::MakeXYWH(50, 50, 250, 250), radii);

    gfx::BoxBorder border;
    border.setBackgroundColor(SK_ColorMAGENTA);
    border.setStyles({ gfx::BorderStyle::kSolid,gfx::BorderStyle::kDouble,gfx::BorderStyle::kDotted,gfx::BorderStyle::kDashed });
    border.setColors({ SK_ColorRED, SK_ColorGREEN, SK_ColorBLUE, SK_ColorBLACK });
    border.setWidths(gfx::Insets::MakeLTRB(10, 20, 30, 40));

    r = border.insets().outset(r);

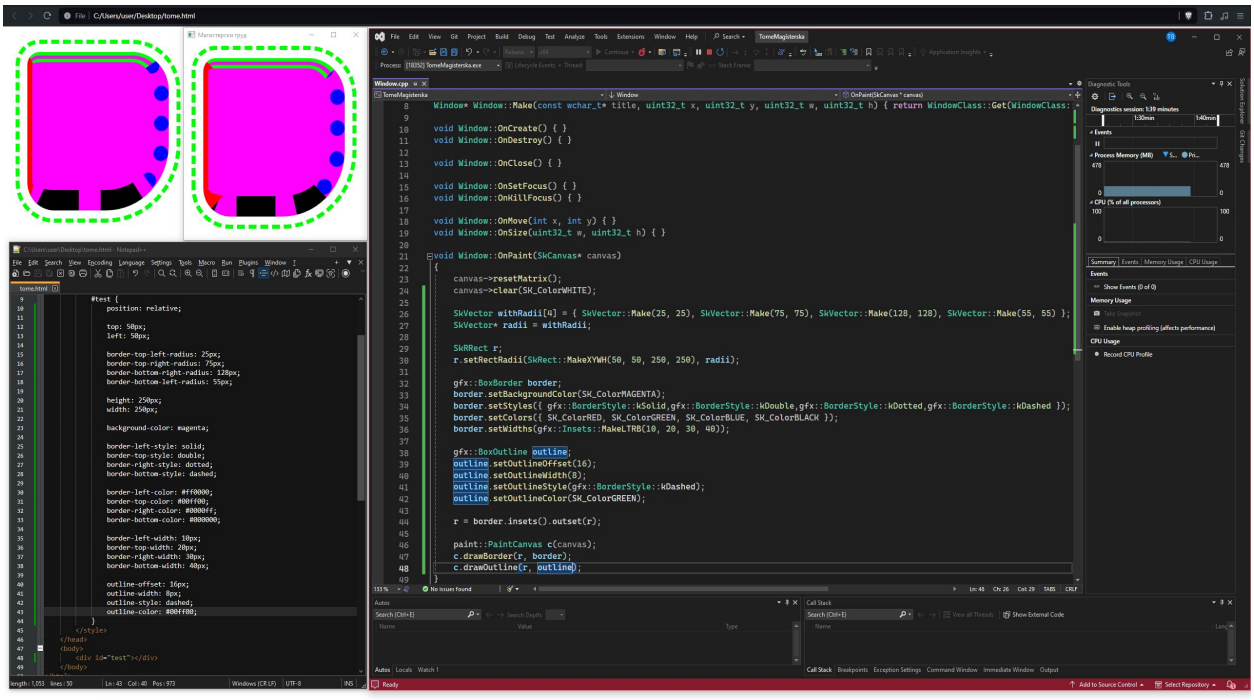
    paint::PaintCanvas c(canvas);
    c.drawBorder(r, border);
}
```

*Слика 47. Изворниот код за цртање на border, outline и box-shadow. Во конкретниот пример, се користи само border*



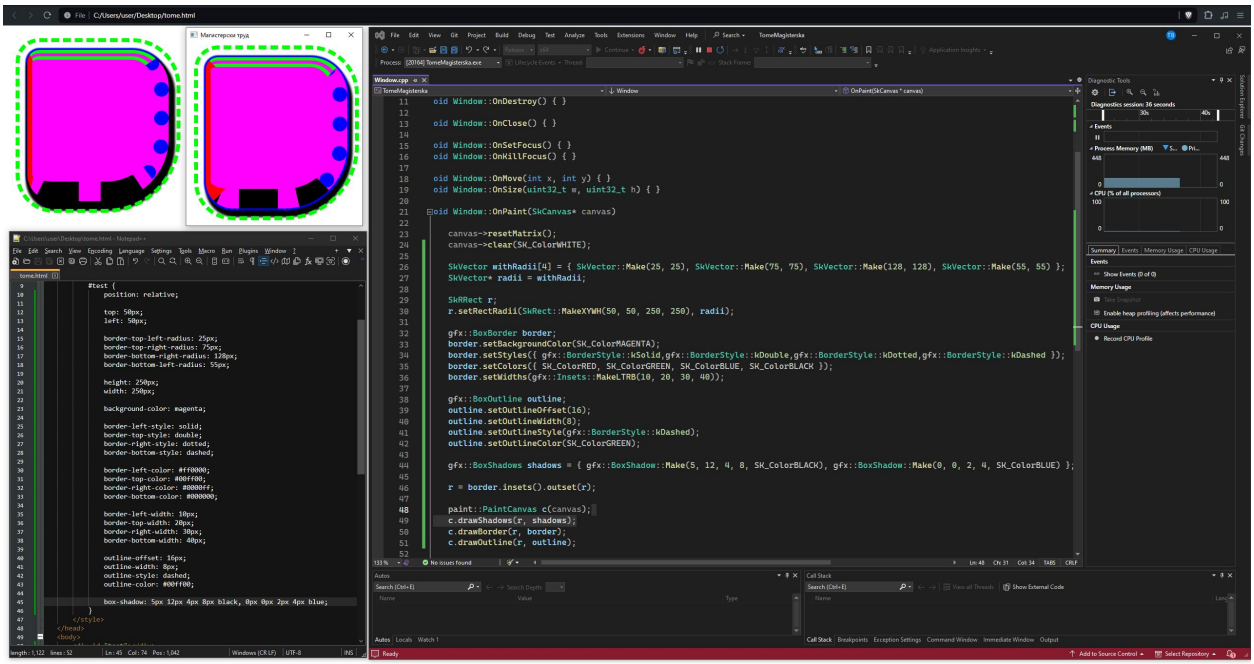
*Слика 48. Демонстрација на CSS border, со сите 4 различни страни*

Следно, ќе се демонстрира на истото што е веќе нацртано, како да се нанесе еден CSS outline.



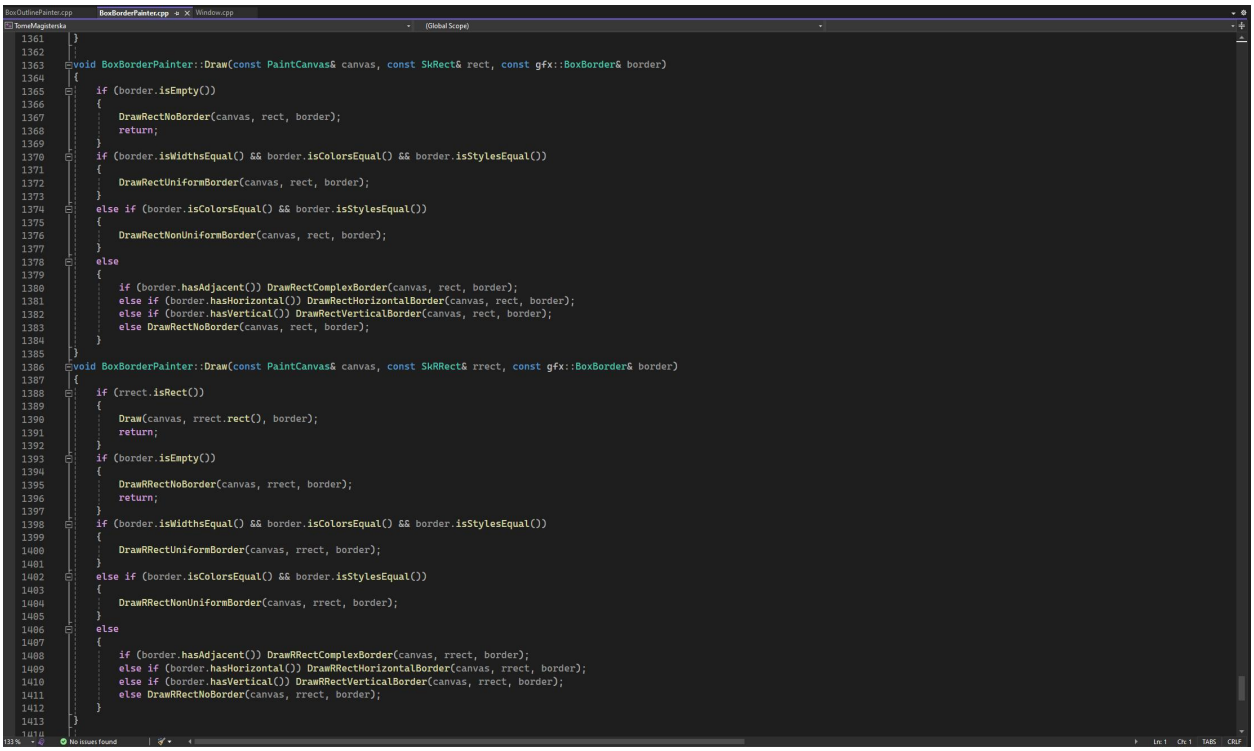
*Слика 49. Демонстрација на CSS outline*

Следно, ќе се демонстрира и box-shadows на истата слика.



*Слика 50. Демонстрација на CSS box-shadows*

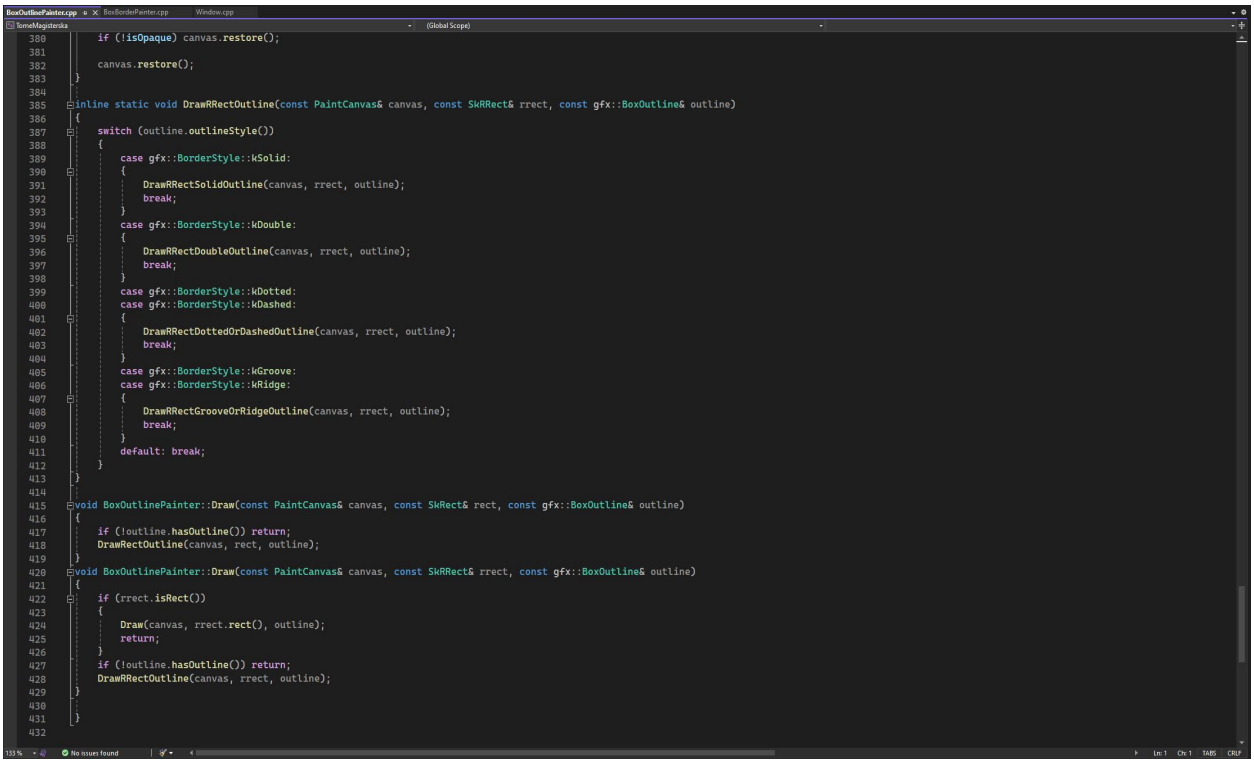
Како што кажавме претходно, кодот за цртање е долго време анализиран, адаптиран и оптимизиран, за потребите на овој труд, односно за framework. Кодот има специјални случаи кога се црта со команди “drawRect”, кога со “drawRRect”, кога со “drawDRRect”, кога со “drawPath”, итн. Исто така, има случаи каде треба да се исече дел од геометријата, дури и неколку пати, со цел да не се нанесат несакани пиксели во несакани простори. И тука има оптимизации кога се сече со “clipRect”, кога се сече со “clipRRect” и кога се сече со “clipPath”. Следно, ќе се прикаже само дел од изворниот код за цртање на border и outline. Изворниот код за box-shadows е многу пократок.



```
}

void BoxBorderPainter::Draw(const PaintCanvas& canvas, const SkRect& rect, const gfx::BoxBorder& border)
{
    if (border.isEmpty())
    {
        DrawRectNoBorder(canvas, rect, border);
        return;
    }
    if (border.isWidthsEqual() && border.isColorsEqual() && border.isStylesEqual())
    {
        DrawRectUniformBorder(canvas, rect, border);
    }
    else if (border.isColorsEqual() && border.isStylesEqual())
    {
        DrawRectNonUniformBorder(canvas, rect, border);
    }
    else
    {
        if (border.hasAdjacent()) DrawRectComplexBorder(canvas, rect, border);
        else if (border.hasHorizontal()) DrawRectHorizontalBorder(canvas, rect, border);
        else if (border.hasVertical()) DrawRectVerticalBorder(canvas, rect, border);
        else DrawRectNoBorder(canvas, rect, border);
    }
}
void BoxBorderPainter::Draw(const PaintCanvas& canvas, const SkRRect& rrect, const gfx::BoxBorder& border)
{
    if (rrect.isRect())
    {
        Draw(canvas, rrect.rect(), border);
        return;
    }
    if (border.isEmpty())
    {
        DrawRRectNoBorder(canvas, rrect, border);
        return;
    }
    if (border.isWidthsEqual() && border.isColorsEqual() && border.isStylesEqual())
    {
        DrawRRectUniformBorder(canvas, rrect, border);
    }
    else if (border.isColorsEqual() && border.isStylesEqual())
    {
        DrawRRectNonUniformBorder(canvas, rrect, border);
    }
    else
    {
        if (border.hasAdjacent()) DrawRRectComplexBorder(canvas, rrect, border);
        else if (border.hasHorizontal()) DrawRRectHorizontalBorder(canvas, rrect, border);
        else if (border.hasVertical()) DrawRRectVerticalBorder(canvas, rrect, border);
        else DrawRRectNoBorder(canvas, rrect, border);
    }
}
```

*Слика 51. Демонстрација на изворниот код за цртање на CSS border. Специфично, ова е почетокот, каде што се одлучува по кој логичен пат треба да се нацрта тој border*



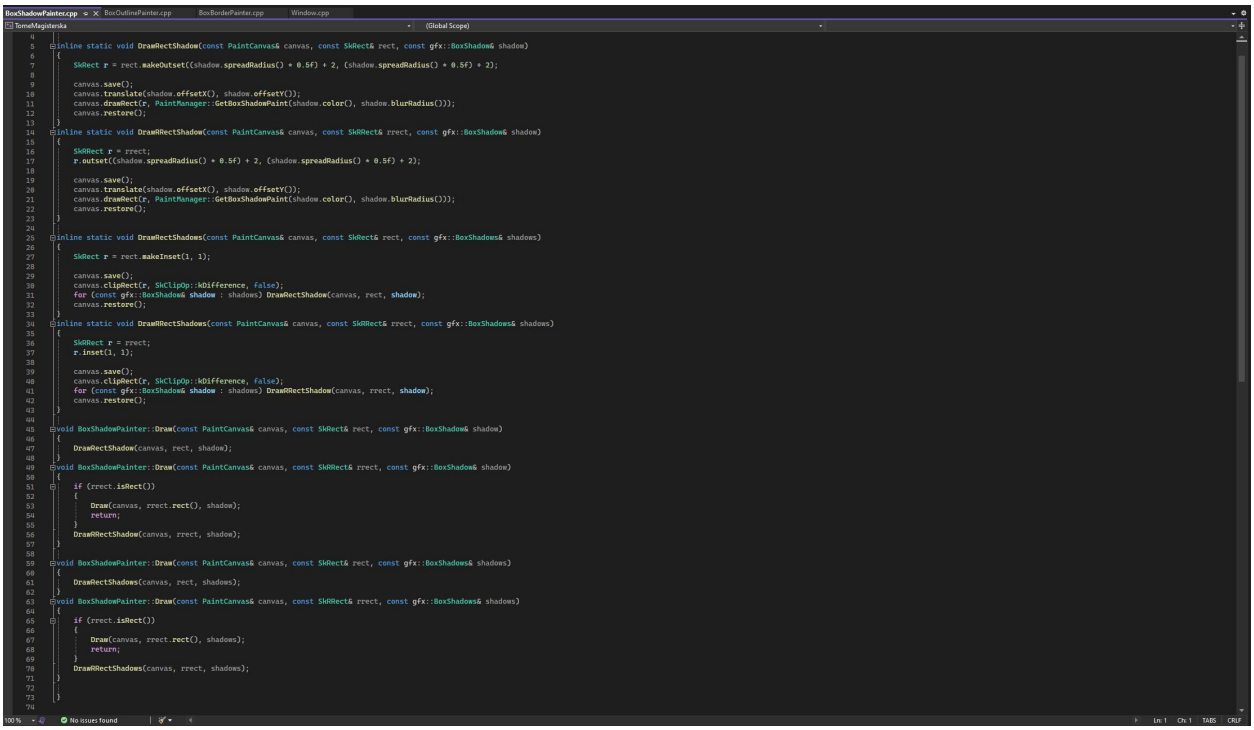
```
    if (!isOpaque) canvas.restore();

    canvas.restore();
}

inline static void DrawRRectOutline(const PaintCanvas& canvas, const SkRRect& rrect, const gfx::BoxOutline& outline)
{
    switch (outline.outlineStyle())
    {
        case gfx::BorderStyle::kSolid:
        {
            DrawRRectSolidOutline(canvas, rrect, outline);
            break;
        }
        case gfx::BorderStyle::kDouble:
        {
            DrawRRectDoubleOutline(canvas, rrect, outline);
            break;
        }
        case gfx::BorderStyle::kDotted:
        case gfx::BorderStyle::kDashed:
        {
            DrawRRectDottedOrDashedOutline(canvas, rrect, outline);
            break;
        }
        case gfx::BorderStyle::kGroove:
        case gfx::BorderStyle::kRidge:
        {
            DrawRRectGrooveOrRidgeOutline(canvas, rrect, outline);
            break;
        }
        default: break;
    }
}

void BoxOutlinePainter::Draw(const PaintCanvas& canvas, const SkRect& rect, const gfx::BoxOutline& outline)
{
    if (!outline.hasOutline()) return;
    DrawRectOutline(canvas, rect, outline);
}
void BoxOutlinePainter::Draw(const PaintCanvas& canvas, const SkRRect& rrect, const gfx::BoxOutline& outline)
{
    if (rrect.isRect())
    {
        Draw(canvas, rrect.rect(), outline);
        return;
    }
    if (!outline.hasOutline()) return;
    DrawRRectOutline(canvas, rrect, outline);
}
```

*Слика 52. Демонстрација на изворниот код за цртање на CSS outline. Специфично, ова е почетокот, каде што се одлучува по кој логичен пат треба да се нацрта тој outline*

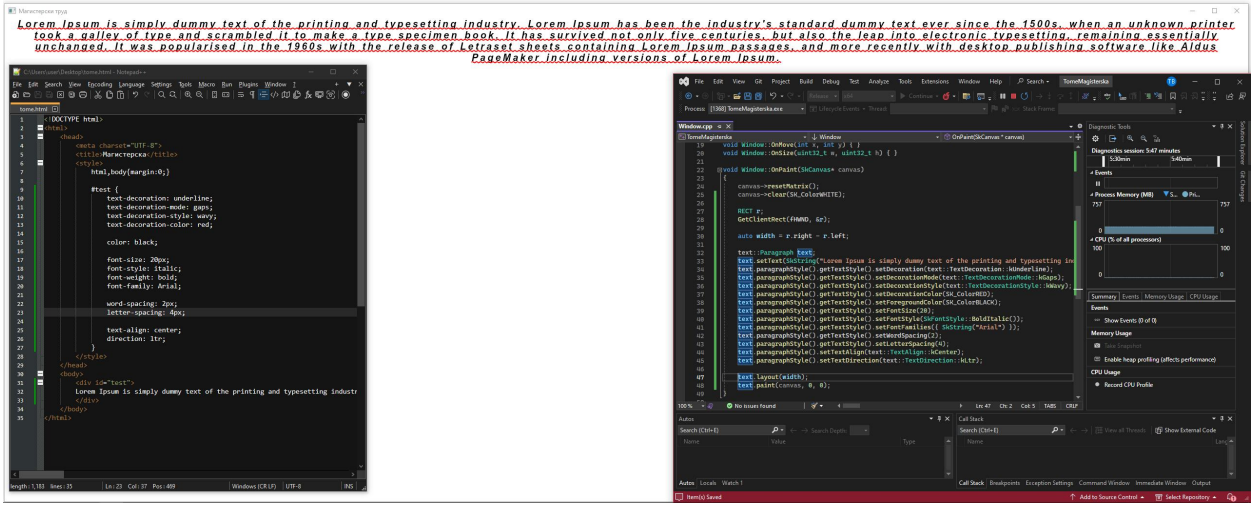


*Слика 53. Демонстрација на изворниот код за цртање на CSS box-shadows*

Следно, ќе се демонстрираат напредни текст операции (пр. обликување на текст, селектирање на текст, CSS својства на текст како text-align, letter-spacing, word-spacing, итн.).

Повеќето од реалниот код за текст е напишан и развиен како модул во Skia од страна на Google. Истиот модул може да се најде во Flutter framework. По име е наречен во Skia SkParagraph. Во нашиот проект, SkParagraph беше преработен за потребите на овој труд. Драстично (дупло) е намалена употребата на меморија од оригиналната класа од Skia. Користење на оваа класа за прикажување на текст е од многу суштински причини. На пример, ако корисникот се реши да кликне еднаш над парче текст, треба тој текст да го позиционира глувчето, односно cursor што се наоѓа над таа буква и да почне индикаторот за внес на текст да „трепка”. Сите GUI апликации го имаат тоа. Втор пример, ако корисникот се реши да кликне двапати над парче текст, треба тој текст да го селектира, на пример, најблискиот „збор” или празен простор во текстот, во зависност каде кликнал двапати. Трет пример, може, ако кликне 3 пати над парче текст, да му се селектира цела линија текст каде кликнал 3 пати. Сите овие сценарија постојат во повеќето GUI апликации, како што се Microsoft Word, сите модерни прелистувачи, Flutter, Qt, итн. Ако треба да се имплементира свое GUI, независно од оперативниот систем, работите

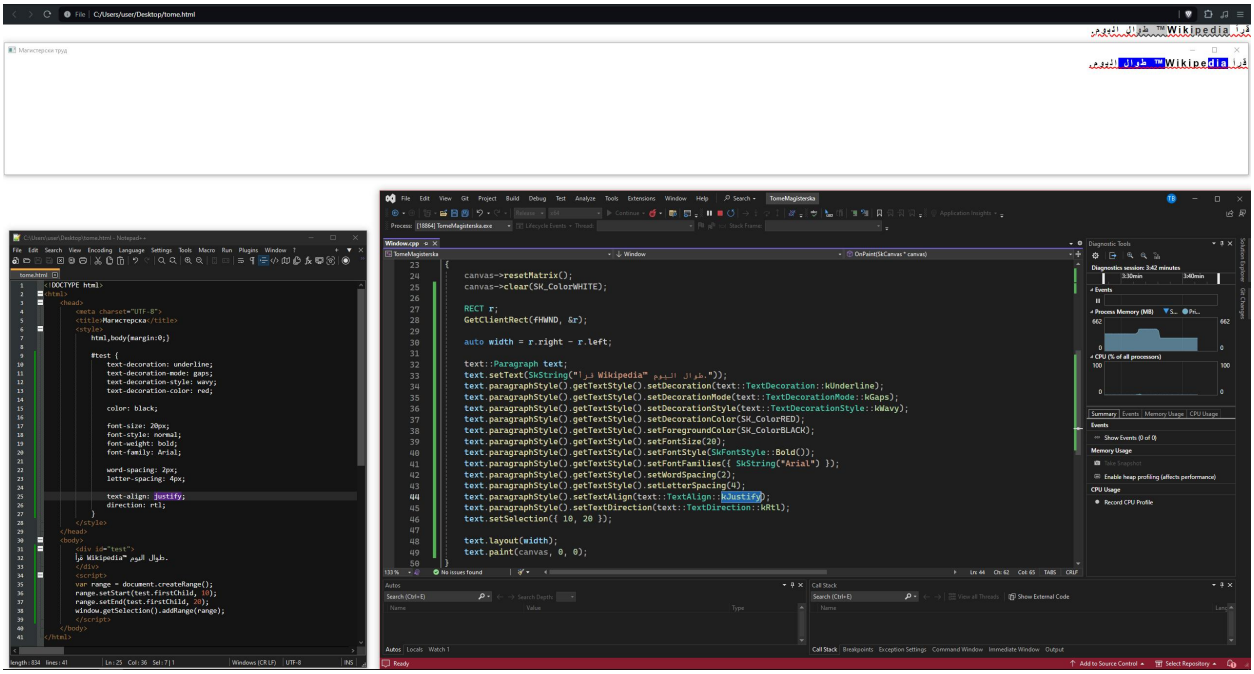
опишани претходно би требало креираниот софтвер да ги поддржува. Други причини зашто класата е од суштинска потреба е заради својства како лево-кон-десно и десно-кон-лево пишување на писмо, калкулација на позициите и големините на самите знаци, односно glyphs во текстот (за да може да се калкулира каде да се „селектира" текстот, но и да се одлучи над кој знак/буква се кликнало), text-align CSS својство (дури и Justify), како и некои слични декорации на текст, позадини и сенки.




*Слика 54. Демонстрација на цртање на текст како прелистувачот Chromium*

Следен пример е цртање на десно-кон-лево писмо.

Текстот "اقرأ Wikipedia™ طوال اليوم." може да се напише наопачки, од десно-кон-лево, во прелистувачот, тоа се прави со CSS својство direction: rtl. Истото ќе биде демонстрирано подолу. Дополнително, ќе се направи и селекција на текстот од индекс 10-ти до 20-ти.



*Слика 55. Демонстрација на цртање на текст како прелистувачот Chromium, користејќи десно-кон-лево писмо*

Како што може да се забележи, повеќето CSS својства за цртање се или веќе имплементирани или со малку напор се имплементираат лесно, користејќи ја Skia. Но, што ако сакаме нешто повеќе од само цртање? Може да се почне да се гради сопствен UI framework со овие алатки кои досега се презентираа. Но, тоа би било многу лимитирачки, со оглед на тоа дека може да се направат само основни копчиња со овие елементи, икони и текстуални елементи, но не таму каде што треба да се позиционираат. Со други зборови, елементите во прилагоденото дрво на сопствениот framework не би знаеле каде да се позиционираат, ниту колкава големина ќе им треба. За тоа, потребен е друг систем, auto layout систем, каде елементите сами, или со помош на експлицитно дефинирани големини, минимални и/или максимални, релативни/апсолутни позиции, margins, borders, paddings, итн. ќе се позиционираат на екранот, релативно од нивниот предок, односно родител во дрвото.

При истражување на таков систем, постои библиотека што го прави тоа и се одлучив да ја искористам библиотеката, која се вика ***yoga,*** креирана од Meta/Facebook. Во рамките на овој магистерски труд, оваа библиотека е исто така проучена и вградена во рамките на нашето дрво за вршење layout и понатаму и цртање. Исто така, како и SkParagraph, така и оваа библиотека, е драстично променета за да функционира и работи

на специфичен начин, така што ни дозволува да ја прошируваме, користиме и менуваме на каков било  начин. На пример, во оригиналната библиотека, секој јазол знае да има свој подјазол . Но, ова не секогаш има смисла. Текст или слика не би можеле да имаат свој подјазли и ова е една од многуте причини зашто е направена одлуката да се вметне библиотека во проектот и драстично да се менува, после анализа на оригиналниот изворен код.

Библиотеката е намената воглавно за да се вградува во други проекти. Самата библиотека имплементира Flexbox layout систем, сличен на тој каков што е во прелистувачите. Библиотеката, исто така, се користи од страна на React Native.

Единствената промена којашто треба да се направи на проектот е да се промени C++ верзијата од C++17 во C++20. Поради тоа што е под наша контрола цела библиотека, ова може и да го менуваме лесно, но со малку напор.

Да се разгледа HTML дрвото и неговите CSS својства кои сакаме да ги прикажеме на екран:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Магистерска</title>
    <style>
      html,body{margin:0;}

      #root {
        display: flex;

        padding: 80px 100px;

        flex-direction: row;
        align-items: center;
      }

      #r {
        display: flex;

        background-color: #0000ff7f;

        border-radius: 24px;

        min-width: 300px;
        min-height: 300px;

        padding: 10px;

        border-left-width: 10px;
        border-top-width: 20px;
        border-right-width: 30px;
        border-bottom-width: 40px;

        border-left-style: solid;
        border-top-style: double;
        border-right-style: dotted;
        border-bottom-style: dashed;

        border-left-color: magenta;
```

```
            border-top-color: black;
            border-right-color: #00ff00;
            border-bottom-color: cyan;

            outline-width: 10px;
            outline-style: dashed;
            outline-color: #00ff00;
            outline-offset: 25px;

            box-shadow: 10px 0px 10px 5px magenta;

            justify-content: center;
            flex-shrink: 1;
            flex-grow: 1;

            color: white;
            font-family: Arial;
            font-size: 15px;
            font-weight: bold;
            text-align: center;
        }
        #r2 {
            display: flex;

            background-color: #00ff007f;

            border-radius: 240px;

            padding: 20px 40px;

            border-left-width: 10px;
            border-top-width: 20px;
            border-right-width: 30px;
            border-bottom-width: 40px;

            margin: 0px 80px;

            border-left-style: solid;
            border-top-style: double;
            border-right-style: dotted;
            border-bottom-style: dashed;

            border-left-color: magenta;
            border-top-color: black;
            border-right-color: #00ff00;
            border-bottom-color: cyan;

            outline-width: 5px;
            outline-style: double;
            outline-color: #ff0000;
            outline-offset: 15px;

            box-shadow: -2px -2px 24px 24px blue;

            justify-content: center;
            flex-shrink: 1;
            flex-grow: 1;

            text-decoration-style: dashed;
            text-decoration: overline;
            text-decoration-color: red;
            color: black;
            font-family: Arial;
            font-size: 22px;
        }
        #slika {
            display: flex;

            max-height: 400px;
            max-width: 400px;
        }
    </style>
</head>
<body>
    <div id="root">
```

```
        <div id="r">Button</div>
        <div id="r2">Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.</div>
        <img id="slika" src="slika.jpg">
    </div>
    <script>
    var range = document.createRange();
    range.setStart(r2.firstChild, 58);
    range.setEnd(r2.firstChild, 254);
    window.getSelection().addRange(range);
    </script>
  </body>
</html>
```

Каде што имаме и CSS својства за визуелен приказ, сега постојат и CSS својства кои се дел од layout, а не од paint фазата.

Во приказот на дрвото подолу, се прави автоматска споредба на CSS својствата. Може да се забележи дека истите параметри се користат и на крај се добива ист резултат.

```
void Window::OnCreate()
{
    fTree.setRoot(fTree.create<yoga::Box>());

    fTree.root()->style().setPadding(yoga::Edge::Left, yoga::value::points(100));
    fTree.root()->style().setPadding(yoga::Edge::Top, yoga::value::points(80));
    fTree.root()->style().setPadding(yoga::Edge::Right, yoga::value::points(100));
    fTree.root()->style().setPadding(yoga::Edge::Bottom, yoga::value::points(80));
    fTree.root()->style().setFlexDirection(yoga::FlexDirection::Row);
    fTree.root()->style().setAlignItems(yoga::Align::Center);

    auto r = fTree.create<yoga::Box>();
    r->setBackgroundColor(0x7ff0000ff);
    r->radii().set(24, 24, 24, 24);
    r->style().setMinDimension(yoga::Dimension::Width, yoga::value::points(300));
    r->style().setMinDimension(yoga::Dimension::Height, yoga::value::points(300));
    r->style().setPadding(yoga::Edge::Left, yoga::value::points(10));
    r->style().setPadding(yoga::Edge::Top, yoga::value::points(10));
    r->style().setPadding(yoga::Edge::Right, yoga::value::points(10));
    r->style().setPadding(yoga::Edge::Bottom, yoga::value::points(10));
    r->style().setBorder(yoga::Edge::Left, yoga::value::points(10));
    r->style().setBorder(yoga::Edge::Top, yoga::value::points(20));
    r->style().setBorder(yoga::Edge::Right, yoga::value::points(30));
    r->style().setBorder(yoga::Edge::Bottom, yoga::value::points(40));
    r->setBorderStyle(gfx::BoxSide::kLeft, gfx::BorderStyle::kSolid);
    r->setBorderStyle(gfx::BoxSide::kTop, gfx::BorderStyle::kDouble);
    r->setBorderStyle(gfx::BoxSide::kRight, gfx::BorderStyle::kDotted);
    r->setBorderStyle(gfx::BoxSide::kBottom, gfx::BorderStyle::kDashed);
    r->setBorderColor(gfx::BoxSide::kLeft, SK_ColorMAGENTA);
    r->setBorderColor(gfx::BoxSide::kTop, SK_ColorBLACK);
    r->setBorderColor(gfx::BoxSide::kRight, SK_ColorGREEN);
    r->setBorderColor(gfx::BoxSide::kBottom, SK_ColorCYAN);
    r->outline().set(10, gfx::BorderStyle::kDashed, SK_ColorGREEN, 25);
    r->addBoxShadow(gfx::BoxShadow::Make(10, 0, 10, 5, SK_ColorMAGENTA));
    r->style().setJustifyContent(yoga::Justify::Center);
    r->style().setFlexShrink(yoga::FloatOptional(1.0f));
    r->style().setFlexGrow(yoga::FloatOptional(1.0f));

    auto text = fTree.create<yoga::Text>();
    text->setText(SkString("Button"));
```

```
    text->textStyle().getTextStyle().setForegroundColor(SK_ColorWHITE);
    text->textStyle().getTextStyle().setFontFamilies({ SkString("Arial") });
    text->textStyle().getTextStyle().setFontSize(15);
    text->textStyle().getTextStyle().setFontStyle(SkFontStyle::Bold());
    text->textStyle().setTextAlign(text::TextAlign::kCenter);

    r->addChild(text);

    auto r2 = fTree.create<yoga::Box>();
    r2->setBackgroundColor(0x7f00ff00);
    r2->radii().set(240, 240, 240, 240);
    r2->style().setPadding(yoga::Edge::Left, yoga::value::points(40));
    r2->style().setPadding(yoga::Edge::Top, yoga::value::points(20));
    r2->style().setPadding(yoga::Edge::Right, yoga::value::points(40));
    r2->style().setPadding(yoga::Edge::Bottom, yoga::value::points(20));
    r2->style().setBorder(yoga::Edge::Left, yoga::value::points(10));
    r2->style().setBorder(yoga::Edge::Top, yoga::value::points(20));
    r2->style().setBorder(yoga::Edge::Right, yoga::value::points(30));
    r2->style().setBorder(yoga::Edge::Bottom, yoga::value::points(40));
    r2->style().setMargin(yoga::Edge::Left, yoga::value::points(80));
    r2->style().setMargin(yoga::Edge::Right, yoga::value::points(80));
    r2->setBorderStyle(gfx::BoxSide::kLeft, gfx::BorderStyle::kSolid);
    r2->setBorderStyle(gfx::BoxSide::kTop, gfx::BorderStyle::kDouble);
    r2->setBorderStyle(gfx::BoxSide::kRight, gfx::BorderStyle::kDotted);
    r2->setBorderStyle(gfx::BoxSide::kBottom, gfx::BorderStyle::kDashed);
    r2->setBorderColor(gfx::BoxSide::kLeft, SK_ColorMAGENTA);
    r2->setBorderColor(gfx::BoxSide::kTop, SK_ColorBLACK);
    r2->setBorderColor(gfx::BoxSide::kRight, SK_ColorGREEN);
    r2->setBorderColor(gfx::BoxSide::kBottom, SK_ColorCYAN);
    r2->outline().set(5, gfx::BorderStyle::kDouble, SK_ColorRED, 15);
    r2->addBoxShadow(gfx::BoxShadow::Make(-2, -2, 24, 24, SK_ColorBLUE));
    r2->style().setJustifyContent(yoga::Justify::Center);
    r2->style().setFlexShrink(yoga::FloatOptional(1.0f));
    r2->style().setFlexGrow(yoga::FloatOptional(1.0f));

    auto text2 = fTree.create<yoga::Text>();
    text2->setText(SkString("Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an
unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially
unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker
including versions of Lorem Ipsum."));
    text2->textStyle().getTextStyle().setDecorationStyle(text::TextDecorationStyle::kDashed);
    text2->textStyle().getTextStyle().setDecoration(text::TextDecoration::kOverline);
    text2->textStyle().getTextStyle().setDecorationColor(SK_ColorRED);
    text2->textStyle().getTextStyle().setForegroundColor(SK_ColorBLACK);
    text2->textStyle().getTextStyle().setFontFamilies({ SkString("Arial") });
    text2->textStyle().getTextStyle().setFontSize(22);
    text2->textStyle().getTextStyle().setFontStyle(SkFontStyle::Normal());
    text2->setSelection(text::TextRange(58, 254));

    r2->addChild(text2);

    auto slika = fTree.create<yoga::Image>();
    slika->style().setMaxDimension(yoga::Dimension::Height, yoga::value::points(400));
    slika->style().setMaxDimension(yoga::Dimension::Width, yoga::value::points(400));
    slika->loadImageFromFile("C:/Users/user/Desktop/slika.jpg");

    fTree.root()->addChild(r);
    fTree.root()->addChild(r2);
    fTree.root()->addChild(slika);
}
```

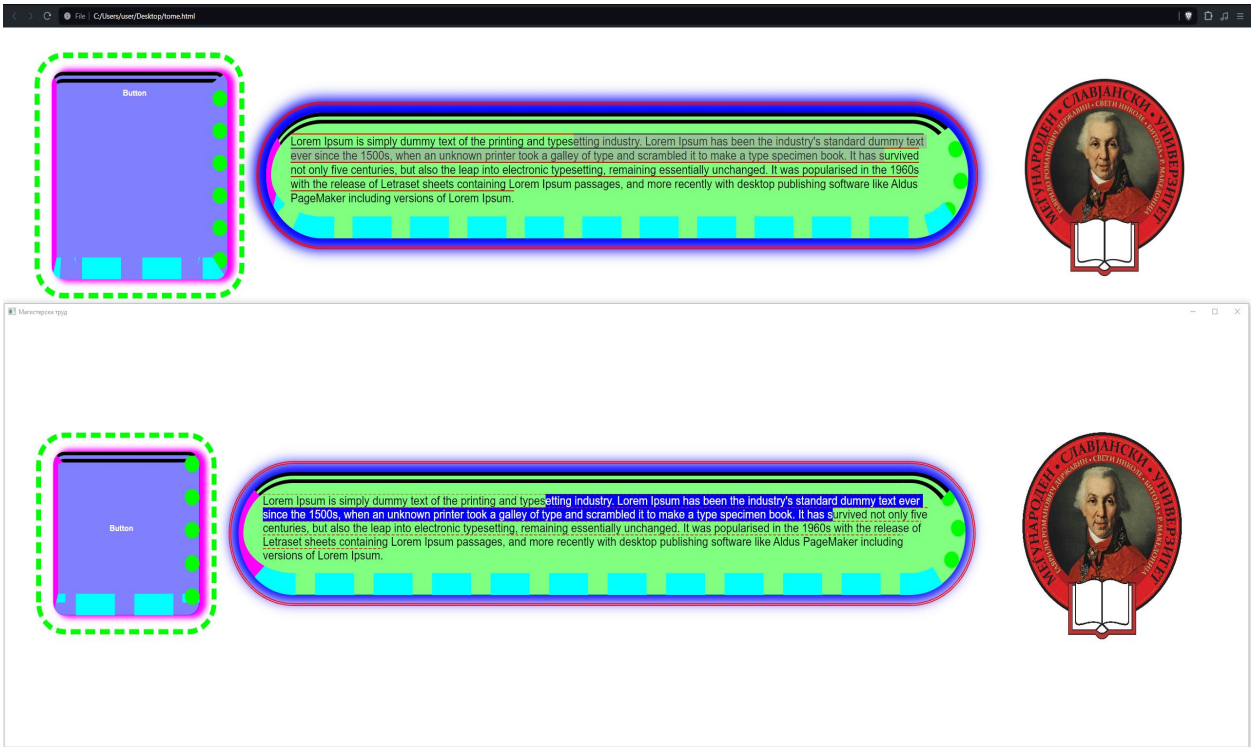
Затоа што ова се извршува кога ќе се креира прозорецот, треба да му се каже на GUI дрвото да направи relayout на своите елементи повторно кога на прозорецот ќе му се сменат димензиите, како широчината или висината. Исто така, на OnPaint методот, ќе се повика методот за цртање на GUI дрвото.

```
void Window::OnSize(uint32_t w, uint32_t h)
{
    RECT size_rect;
    GetClientRect(fHWND, &size_rect);

    fTree.layout(size_rect.right - size_rect.left, size_rect.bottom - size_rect.top);
}

void Window::OnPaint(SkCanvas* canvas)
{
    canvas->resetMatrix();
    canvas->clear(SK_ColorWHITE);
    fTree.draw(canvas);
}
```

Резултатот на ова е GUI дрво кој знае да направи layout и да се нацрта сам себе, користејќи ја библиотеката Skia, преку класата SkCanvas.




*Слика 56. Демонстрација на Flexbox layout користејќи ја библиотеката Yoga и цртање на позициите и големините пресметани од Yoga преку Skia. Горната слика претставува output од прелистувач користејќи HTML и CSS, додека долниот прозорец е наша имплементација на CSS својства преку C++*

До овој момент, GUI елементите уште не знаат и немаат поим што е глувче, тастатура, настан, справувач со настан (event handler), итн., сè додека е возможно овие работи да се имплементираат, независно од оперативниот систем.

Прилагоденото GUI дрво ја користи библиотеката Yoga за да пресмета layout и токму тоа се случува кога ќе се повика layout методот, како што е прикажано погоре. Да видиме како се поминува дрвото за да се нацрта цело дрво.

```
void Node::traverse(const std::function<void(Node*)>& startCallback, const std::function<void(Node*)>& endCallback, const std::function<bool(const Node*)>& condCallback)
{
    if (!condCallback(this)) return;

    startCallback(this);

    if (Is<Box>())
    {
        Node* curr = As<Box>()->getFirstChild();
        while (curr)
        {
            bool cond = condCallback(curr);
            if (cond) startCallback(curr);
            if (cond && curr->getChildCount())
            {
                curr = curr->As<Box>()->getFirstChild();
            }
            else
            {
                if (cond) endCallback(curr);
                if (curr->getNextSibling()) curr = curr->getNextSibling();
                else
                {
                    curr = curr->getParent();
                    while (curr)
                    {
                        endCallback(curr);
                        if (curr == this) return;
                        else if (curr->getNextSibling())
                        {
                            curr = curr->getNextSibling();
                            break;
                        }
                        else
                        {
                            curr = curr->getParent();
                        }
                    }
                }
            }
        }
    }
    else
    {
        endCallback(this);
    }
}

} // namespace yoga
```

*Слика 57. Демонстрација на изминување на едно дрво за цртање.*

За едно дадено дрво, на пример:

```
<root>
  <one>
    <inner/>
  </one>
  <two/>
  <three/>
</root>
```

Кодот погоре ќе резултира со следниот резултат:

startCallback – root

startCallback – one

startCallback – inner

endCallback – inner

endCallback – one

startCallback – two

endCallback – two

startCallback – three

endCallback – three

endCallback - root

Со ваквото изминување на едно дрво, на нерекурзивен начин, се дозволува многу брзо и ефикасно да се помине целото дрво, без разлика колку е длабоко, колку елементи има, итн., на точен начин, како што бараат некои ефекти и/или феномени. Еден пример зашто е ова изминување на дрво е суштинско, е имплементација на CSS својства како transform, filter и opacity. На пример, ако се нанесе opacity од 0.5 на root, тогаш сите елементи треба да имаат боја со алфа канал максимум од 0.5. Ако некој поделемент има дополнителен opacity, тогаш финалниот алфа канал за тој елемент и неговите поделементи би биле 0.25 и би се нацртале коректно на екран, како и другите негови поделементи, како во повеќето други софтвери.

Следно, за да може дрвото да биде некогаш корисно, потребно е откога ќе се изврши layout и paint, да може да се кликне на некој елементи на екранот за да нешто се случи. Дрвото, како што е поставено со елементи, има свои позиции, големини, но и матрици. Кога се проверува дали во некоја точка, односно координата лежат елементи, потребен е контекст сличен како SkCanvas, каде што може да се направи save(), restore(), concat(matrix), translate(x, y), итн., за да може коректно да се движиме низ дрвото. Финалната матрица се користи заедно со точката, односно координатата за да се трансформира секоја геометрија која треба да се тестира со таа точка. Зависно од тоа дали матрицата ги прави правите агли да не се прави после трансформацијата, мора да се искористи формула различна од x >= Left && y >= Top && x <= Right && y <= Bottom.

Подолу ќе се прикаже код за “hit-testing”:

```
#pragma once

#include <skia/include/core/SkM44.h>

#include <vector>

class HitTestContext
{
public:
    HitTestContext(SkPoint point) : fPoint(std::move(point)) { fRecs.reserve(16); }

public:
    constexpr const SkPoint& point() const { return fPoint; }
    constexpr const SkMatrix& matrix() const { return fCurrRec.fMatrix; }

public:
    void translate(SkScalar x, SkScalar y) { concat(SkMatrix::Translate(x, y)); }

    void concat(const SkMatrix& matrix) { fCurrRec.fMatrix.postConcat(matrix); }

public:
    void save() { fRecs.emplace_back(fCurrRec); }
    void restore()
    {
        if (fRecs.empty()) return;
        fCurrRec = fRecs.back();
        fRecs.pop_back();
    }

private:
    struct Rec
    {
        SkMatrix fMatrix;
    };

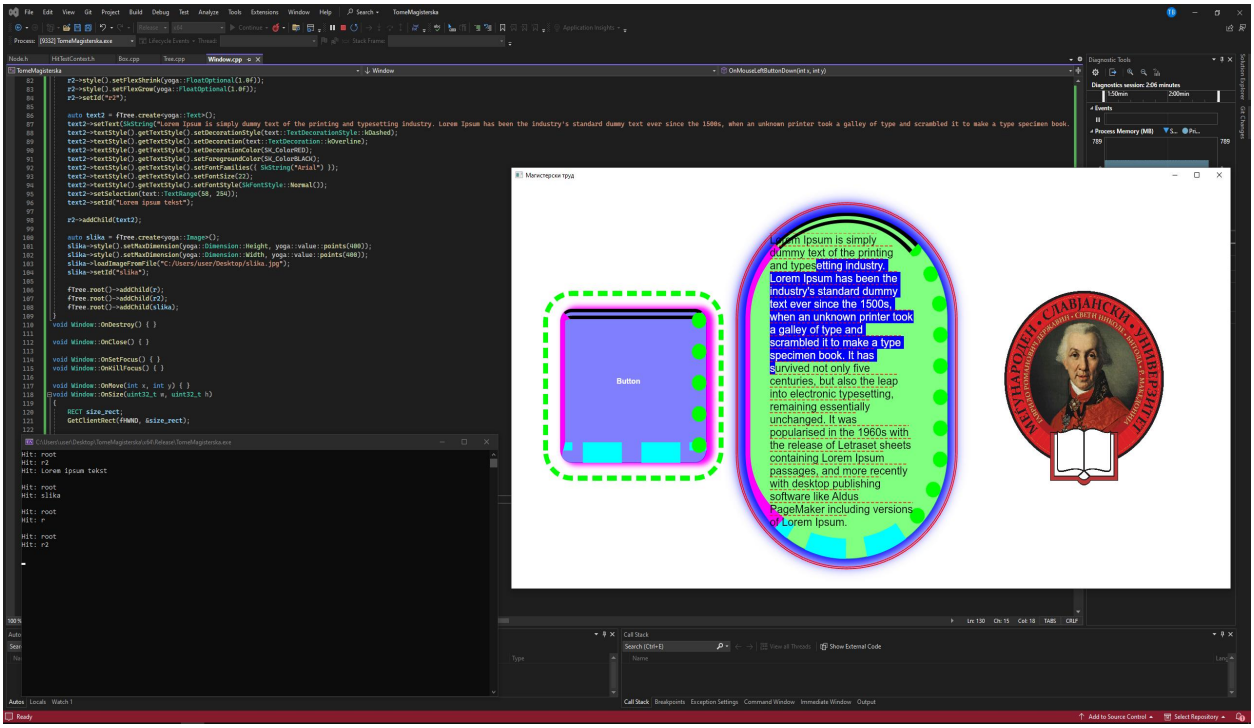
    SkPoint fPoint;
    Rec fCurrRec;
    std::vector<Rec> fRecs;
};
```

```
void Window::OnMouseLeftButtonDown(int x, int y)
{
    auto hits = fTree.hitTest(SkPoint::Make(x, y));
    printf("Hits: %d\n", (int)hits.size());
}
std::vector<Node*> Tree::hitTest(const SkPoint& point) const {
    if (!root()) return {};
    HitTestContext ctx(point);
    std::vector<Node*> hits;
    root()->traverse(
        [&ctx, &hits](Node* node)
        {
            ctx.save();
            if (node->OnHitTest(ctx)) hits.emplace_back(node);
        },
        [&ctx](Node* node)
```

```
        {
            ctx.restore();
        },
        [](const Node* node) { return node->isVisible(); }
    );
    return hits;
}
bool Box::OnHitTest(HitTestContext& ctx) const
{
    bool hit;
    if (ctx.matrix().rectStaysRect()) {
        SkRRect r;
        borderRRect().transform(ctx.matrix(), &r);
        hit = r.contains(SkRect::MakeLTRB(ctx.point().x() - 0.5f, ctx.point().y() - 0.5f, ctx.point().x() + 0.5f, ctx.point().y() + 0.5f));
    }
    else {
        SkPath p = SkPath::RRect(borderRRect()).transform(ctx.matrix());
        hit = p.contains(ctx.point().x(), ctx.point().y());
    }
    if (hit && hasTransform()) {
        ctx.concat(transform());
    }
    return hit;
}
```

Листата која се пресметува и која се враќа назад ги чува елементите, на таков начин, што елементите на крајот на таа листа се елементите коишто се најблиску до корисникот, односно окото. Елементите коишто се први во листата се „најдалеку" од окото, во смисла, тие елементи се „зад" други елементи. На пример, root елементот, тој што ги содржи сите во дрвото секогаш ќе биде прв на листата.

За да се направи еден "hit-test", на самите елементи им се додадени тагови, односно "id" за да се идентификуваат еден од друг, за да се знае кои елементи се кликнати на екран. Следната слика ја покажува листата испечатена на екранот и  кои елементи биле кликнати. Со информациите, односно елементите вратени од таа листа, може многу работи да се имплементираат, како на пример, настани. Со тоа, можат апликациите да станат интерактивни.




*Слика 58. Демонстрација на "hit-testing" на едно дрво*

Следно, откога ќе се овозможат нашите елементи да станат интерактивни, следен природен чекор е да се размислува за анимации.

Но, тоа е малку покомплицирано да се имплементира и е надвор од истражувањето претставено во овој труд. Но, генерално, за да се имплементира еден систем за анимации, треба да се размисли тоа што е анимација и како истата анимација да се движи со некое дадено поминато време за да може да се видат вредностите кои на самата крива за анимација или на низата на frames се менуваат преку некое дадено определено време. На пример, да го земеме CSS својството *transition* во предвид. За да се дефинира анимација, односно транзиција да се случи преку некое дадено определено време, синтаксата е следна: *transition: width 1s linear;*. Со тоа, CSS својството *width*, би требало, кога ќе добие нова вредност, да се анимира до новата вредност, од моменталната широчина на елементот, преку 1 секунда време, со линеарна интерполација. Едноставно е кажано, но самата имплементација на таков систем е тешко изведена.

На пример, главната нишка, каде се обработува Windows Message Loop, којашто се справува со настани и кои ги делегира истите настани кон прозорите е врзана и „заглавена" за таа задача да ја извршува, најбрзо што може, кога може. Со тоа кажано, има и некои примери каде Windows знае да влезе во интерен Message Loop и тој знае да не

врати резултат сѐ додека Windows не се справи со него. Во тоа време, главниот Windows Message Loop не работи и едноставно чека Windows да заврши со неговата интерна задача.

Со други зборови, софтверот не може од главната нишка да ги делегира и да се справува со настани за анимации, односно да ги движи анимациите. Ќе мора да се креираат нови нишки и главната задача на тие нишки би била да ги движат тие анимации. Ова е само еден наивен начин како би се справиле со движење на свои прилагодени анимации.

Но, постои друг проблем ако така се одлучи да се движат анимациите. За среќа, има решение за тој проблем. Да се разгледа следниот код:

```
auto start = std::chrono::high_resolution_clock::now();
while (!fThreadStopped)
{
   auto end = std::chrono::high_resolution_clock::now();
   auto t = std::chrono::duration_cast<std::chrono::duration<float, std::milli>>(end - start).count();
   start = end;

   printf("%f\n", t);
}
```

Ако се изврши следниот код, многу брзо процесорот ќе стигне до 100% искористеност. На екран ќе се печатат вредности од 0 или блиску до 0. Тоа е заради тоа што променливата t е изминатато време од последниот пат кога се извршил кодот и тогаш во моментот кога се извршува истиот код повторно. Ова претставува голем проблем, заради тоа што сите ресурси на компјутерот се користат беспотребно.

Но, не само тоа. Дури и да се повика командата на Windows “Sleep(0)” или “Sleep(1)”, ова ќе го реши проблемот со искористеноста на процесорот, но се појавува и друг проблем. Sleep е неконзистентен, поради тоа што прави многу други работи во позадина и дополнително чека на оперативниот систем за да на таа нишка ѝ даде време да изврши следни инструкции. Ова резултира во тоа што, иако можеби е кажано да се чека 3 милисекунди, да се чека цели 15 милисекунди. Нормално, тоа значи во анимација, да пропуштиме дел од кривата и да не видиме дел од анимацијата. Што ако имаме монитор од 144Hz? Или уште пострашно, 240Hz? Времето за чекање, односно t, односно delta t, се калкулира преку следното: t = 1000/Hz. Тоа значи дека ако користиме 240Hz, дека времето за чекање треба да биде 4.166666666666667 милисекунди. Со функција Sleep, тоа е невозможно да се постигне.

Решението за овој проблем е „вертикална синхронизација”, односно “V-Sync”. Со други зборови, фреквенцијата на while loop, би требало да се синхронизира со процесорот на мониторот. На пример, пред да почне да процесира нова „рамка”, односно frame на екран мониторот, процесорот од страна на таа нишка ќе чека за да тој сигнал биде „испратен” од страна на мониторот. Со тоа, фреквенцијата на while loop, односно нишка се синхронизира со брзината на процесирање на една рамка од страна на мониторот. Ова резултира со while loop чија фреквенција е иста со таа на мониторот, дозволувајќи на точен t за движење на самите анимации, да не се искористува многу процесорот и најбитно од сите, нема да има вишок слики за испраќање на екран за приказ.

Има повеќе различни начини како да се дочека тој сигнал. Еден од тие начини е преку самиот графички API. Кога треба да се презентира нова слика да се испрати кон екран за приказ преку графичко API, обично се повикува функција којашто во нејзиното име го има терминот “SwapBuffers”. Во тој момент, интерно, графичкото API, ако е конфигурирано од страна на повикувачот на функцијата да се чека за сигналот, графичкото API ќе го почека тој сигнал пред да ја испрати сликата на екран за приказ. Но, ова е не секогаш добро да се користи поради тоа што треба да се користи графичкото API за да се дочека тој сигнал, но исто така не е добро заради тоа што резултира со презентација на нова слика, а во случајот, се одвива фаза на конструкција на нова слика и потенцијално нема нова слика за презентирање. Со други зборови, потребен е сигналот да се дочека без да се користи графичкото API.

Друг начин е да се користи, на Windows, Desktop Window Manager (DWM). Но, во поранешни верзии на Windows, оваа компонента може од страна на корисникот да се исклучи и функцијата “DwmFlush”, која го користи DWM за да го почека сигналот за VSync да резултира да не работи и да не чека. Ако потенцијално се наслонува на таа функција, while loop ќе има повторно 100% CPU искористеност.

После долго истражување, преку форум за дискусија, се спомнаа 2 функции на Windows за чекање на тој сигнал (D3DKMTOpenAdapterFromHdc и D3DKMTWaitForVerticalBlankEvent), коишто не се наслонуваат на претходните лимитации со графичкото API или со исклучување на DWM. Со следниот код, може да се движат анимациите со точниот t:

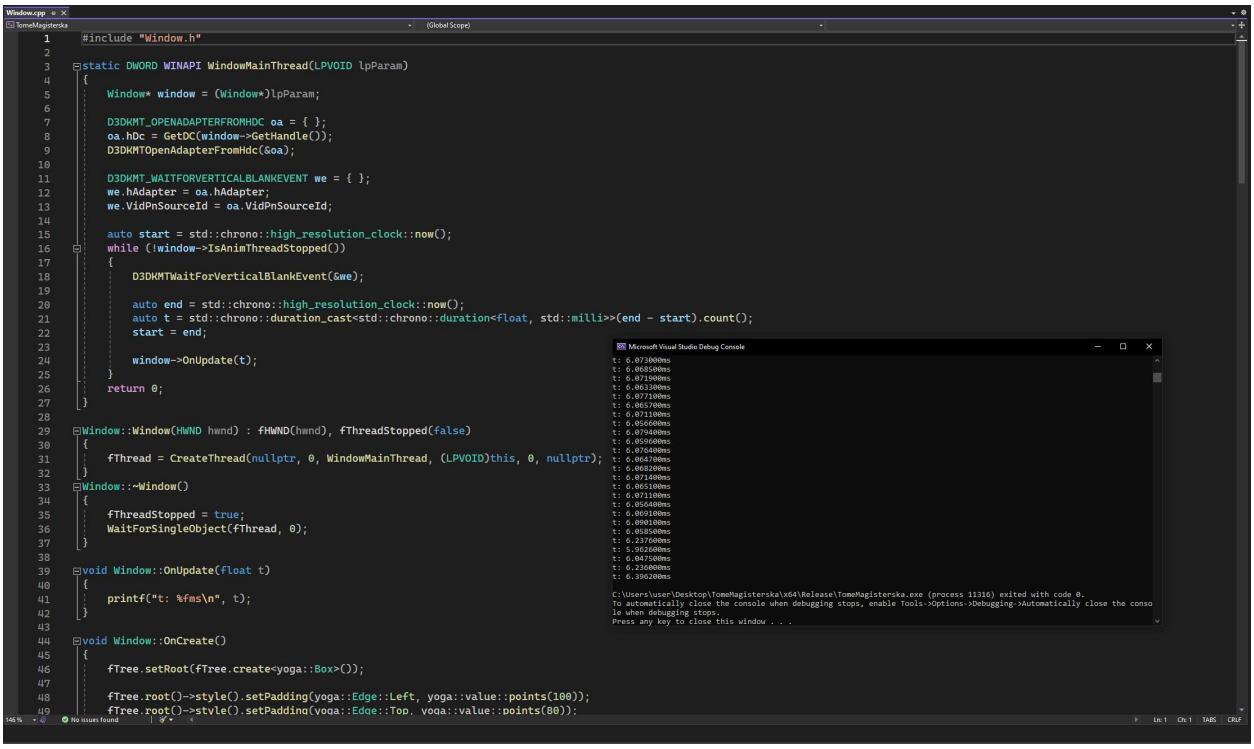
```
static DWORD WINAPI WindowMainThread(LPVOID lpParam)
{
   D3DKMT_OPENADAPTERFROMHDC oa = { };
   oa.hDc = GetDC(HWND_DESKTOP);
   D3DKMTOpenAdapterFromHdc(&oa);

   D3DKMT_WAITFORVERTICALBLANKEVENT we = { };
   we.hAdapter = oa.hAdapter;
   we.VidPnSourceId = oa.VidPnSourceId;

   auto start = std::chrono::high_resolution_clock::now();
   while (!fThreadStopped)
   {
      D3DKMTWaitForVerticalBlankEvent(&we);

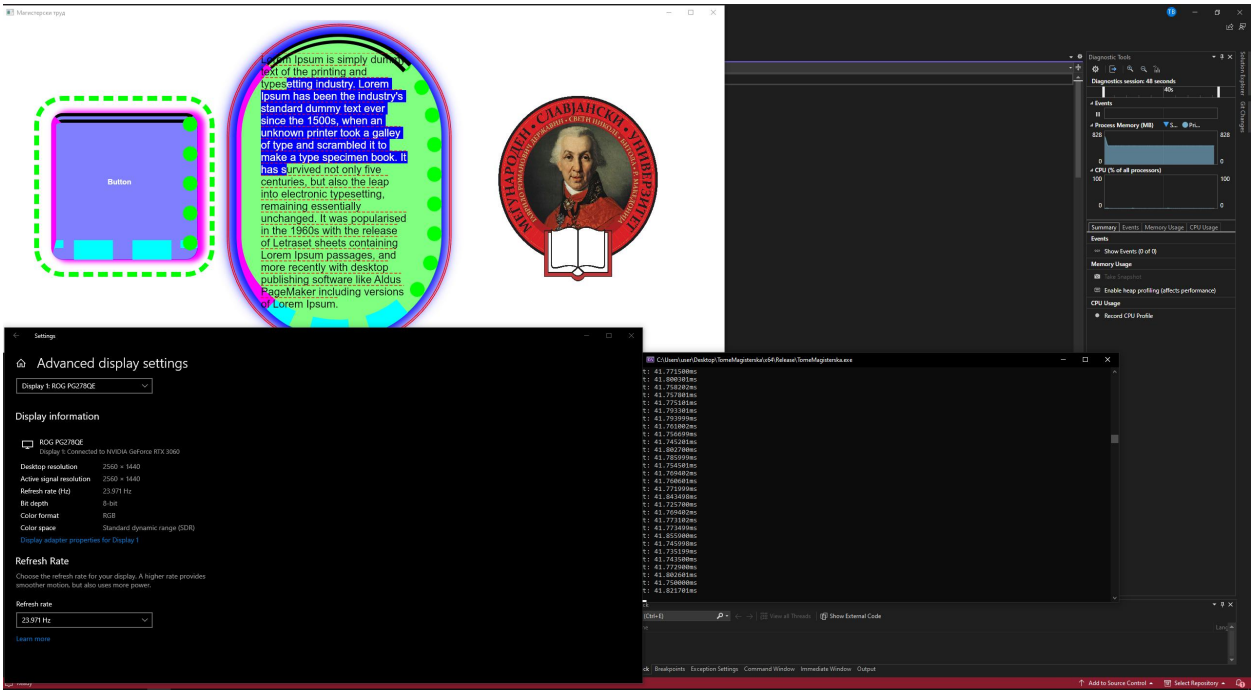
      auto end = std::chrono::high_resolution_clock::now();
      auto t = std::chrono::duration_cast<std::chrono::duration<float, std::milli>>(end - start).count();
      start = end;

      printf("%f\n", t);
   }
   return 0;
}
```




*Слика 59. Демонстрација на чекање на сигналот за "VSync" со 165Hz*

Поради тоа што мониторот на којшто е тестирано ова е од 165Hz, t е приближно еднакво со формулата од претходно: 1000/Hz, односно 1000/165 = 6.06060606. Дури и динамично да се смени фреквенцијата, t во овој случај исто така ќе се смени.

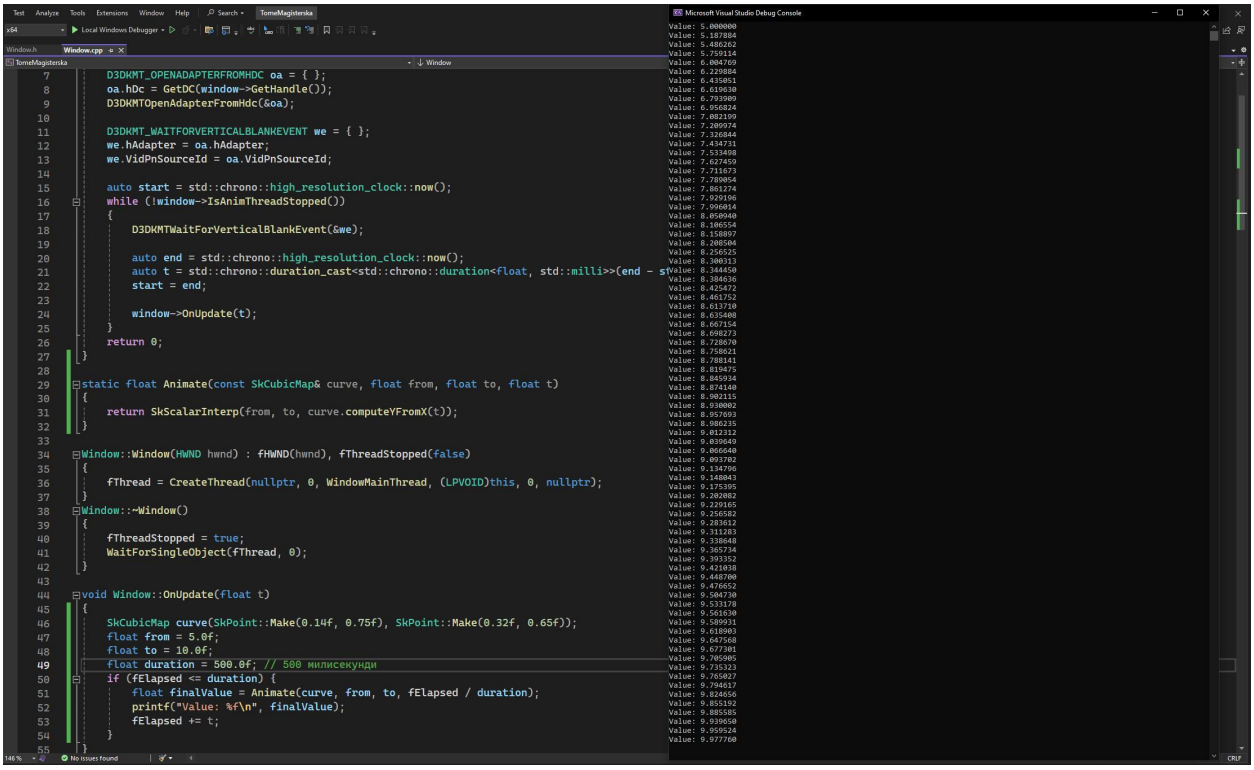



*Слика 60. Демонстрација на чекање на сигналот за "VSync" со 24Hz*

Нормално, ако сакаме интеракција со други објекти надвор од таа нишка од која се чека сигналот за VSync, тогаш ќе треба да се синхронизираат објектите низ различните нишки или спротивно, програмот може и да падне или да не се однесува веќе според очекуваното.

Но, како и да се одлучи дизајнот на системот за анимации и како анимациите ќе се движат, Skia има различни алатки за криви и за пресметување на y според x, односно t. Таа класа се вика SkCubicMap. На сличен начин е дефинирана, така што може сите различни криви и од CSS спецификацијата да се користат како што се linear, ease-in, ease-out, ease-in-out, итн. Се дефинира преку 2 точки, односно SkPoint.

За да може да се дефинира SkCubicMap и да се користи за да се добијат новите вредности од подвижната анимација, обично тоа се прави со следниот код:




*Слика 61. Демонстрација на користење SkCubicMap, крива која може да се користи за да се постигне анимација на екран*

# Заклучок

Целта на овој труд е да се демонстрира еден од многуте начини на кој би можел да се гради свој UI framework, независно од самиот оперативен систем, графичко API или слично. Skia е доста лесна за учење и за применување во креирање на графика. Како што видовме претходно, за да се стигне до специфичен резултат на екран, користење на прелистувачи и различни својства, можно е да се „копираат", односно постигнат истите резултати присутни во поголемите софтвери кога сакаме да имплементираме делови од UI framework кои можеме да ги најдеме во други frameworks. Поради тоа што Skia е тестиран и користен во поголемите frameworks и софтвери, може да се потпреме целосно кон библиотеката и истата да ја користиме во нашиот труд.

Кога треба посложени работи да се имплементираат, како автоматски layout на елементи на екран, може слободно да се искористат други алатки и библиотеки лесно достапни и компатибилни за наши потреби, цели и конфигурации. Тоа го направивме со користење на втора библиотека (којашто беше на крајот вградена и комплетно изменета , повторно, за наши потреби во нашиот UI framework) и со тоа успешно постигнавме, исто така, повторно ист (или сличен) резултат како и во случај на поголемите софтвери.

Во контекст на споменатата анимација при крајот на нашиот практичен дел, во евентуален понатамошен развој на нашиот framework можеме да продолжиме понатаму со креирање на дрвото преку библиотека за вчитување на HTML документ. Еден таков пример е да се користи библиотеката libxml2 и да се интегрира библиотеката во проектот, во смисла да се искористи вчитување на HTML „токени" и да се креира дрвото според резултатите дадени од самата библиотека за секој елемент во HTML дрвото.

Понатаму, може исто така да се искористи или да се креира нов модул за CSS и да се користат Skia објектите и класите (SkScalar, SkMatrix, SkImageFilter, итн.) за да се креираат потребните објекти за да се постигне целта на CSS својството.

Во понатамошните чекори во овој труд, имплементирањето на анимација би можело да се изврши и преку друга нишка, како што беше спомнато погоре. Би можеле да се насочиме кон развој на специфичен анимациски систем.

Понатаму, може да се започне со развивање на настани и со нивниот развој, да се воведе и динамички скриптирачки јазик како JavaScript, Lua или Python и да се овозможи

програмирање на дрвото преку најпознатите светски јазици. Со тоа, прилагодениот UI framework може да стане исто така многу популарен и користен за различни цели.

# Користена литература и линкови


1. „ACM Computing Classification System ToC“. Здружение за компјутерски машини. Преземено на 2024-06-06.
2. Greenberg, Ira (2007). Processing: Creative Coding and Computational Art. Apress. ISBN 978-1-59059-617-3.
3. Винсент, Џејмс (2022). „All these images were generated by Google's latest text-to-image AI“. The Verge. Vox Media. Преземено на 2024-06-06.
4. „Lighting principles for 3D artists from film and art“. GarageFarm. Преземено на 2024-06-06.
5. „LearnOpenGL - Shaders“. Learnopengl.com. Преземено на 2024-06-06.
6. Рот, Роберт Е. (2017). „User Interface and User Experience (UI/UX) Design” Географски информатички науки и технологија Тело на знаење. 2017 година (Q2). doi:10.22224/gistbok/2017.2.5.
7. „Command line vs. GUI“. Computer Hope. Преземено на 2024-06-06.
8. "The GUI versus the Command Line: Which is better? (Part 1)". Операции на Microsoft.com. Microsoft Learn. Преземено на 2024-06-06.
9. "The GUI versus the Command Line: Which is better? (Part 2)". Операции на Microsoft.com. Microsoft Learn. Преземено на 2024-06-06.
10. „Graphical user interface“. ScienceDaily. Преземено на 2024-06-06.
11. Levy, Steven. "Graphical User Interface (GUI)". Britannica.com. Преземено на 2024-06-06.
12. „GUI“. PC Magazine Енциклопедија. pcmag.com. Преземено на 2024-06-06.
13. „chrome“. www.catb.org. Преземено на 2024-06-06.
14. Нилсен, Јакоб (2012). „Browser and GUI Chrome“. Nngroup. Преземено на 2024-06-06.
15. Мартинез, Венди Л. (2011). „Graphical user interfaces: Graphical user interfaces“. Wiley Interdisciplinary Reviews: Computational Statistics. 3 (2): 119–133. doi:10.1002/wics.150. S2CID 60467930.
16. "What is a graphical user interface (GUI)?". IONOS Digitalguide. Преземено на 2024-06-06.

17. „The OpenGL Graphics System: A Specification“ (PDF). 4.0 (Основен профил). Преземено на 2024-06-06.
18. „OpenGL ARB to Pass Control of OpenGL Specification to Khronos Group“. Групата Хронос. Преземено на 2024-06-06.
19. „The Future of OpenGL (forum discussion)“. Групацијата Хронос. Преземено на 2024-06-06.
20. „Khronos News Archives“. Групацијата Хронос. Преземено на 2024-06-06.
21. „Khronos Blog“. Групацијата Хронос. Преземено на 2024-06-06.
22. „Khronos Membership Overview and FAQ“. Khronos.org. Преземено на 2024-06-06.
23. „Khronos OpenGL Registry“. Групацијата Хронос. Преземено на 2024-06-06.
24. „Vulkan“. NVIDIA Developer. Преземено на 2024-06-06.
25. „Vulkan®“. GPUOpen. Преземено на 2024-06-06.
26. „Developing 3D Graphics with Vulkan on Mobile“. Мрежа за програмери на Qualcomm. Преземено на 2024-06-06.
27. Хрушка, Џоел. „Next-generation Vulkan API could be Valve's killer advantage in battling Microsoft“. ExtremeTech. Преземено на 2024-06-06.
28. „Setting Up a Command Structure“. Apple Inc.
29. „Metal Shading Language Guide“. Преземено на 2024-06-06.
30. „Configuring Depth-Stencil Functionality“. Мајкрософт.
31. „RenderStateManager.UseWBuffer“. Мајкрософт.
32. „Texture Blending (Direct3D 9)“. Мајкрософт.
33. „Geometry Blending (Direct3D 9)“. Мајкрософт.
34. „HLSL“. Мајкрософт.
35. „Effects (Direct3D 11)“. Мајкрософт.
36. „Software Rasterizer for DirectX 9.0 SDK“. Преземено на 2024-06-06.
37. „Graphics Pipeline“. Direct3D 11 графика. Microsoft Docs. Преземено на 2024-06-06.
38. „Input-Assembler Stage“. Direct3D 11 графика. Microsoft Docs. Преземено на 2024-06-06.
39. „Vertex Shader Stage“. Direct3D 11 графика. Microsoft Docs. Преземено на 2024-06-06.

40. „Tessellation Stages“. Direct3D 11 графика. Microsoft Docs. Hull-Shader Stage. Преземено на 2024-06-06.
41. „Tessellation Stages“. Direct3D 11 графика. Microsoft Docs. Теселатор сцена. Преземено на 2024-06-06.
42. „Tessellation Stages“. Direct3D 11 графика. Microsoft Docs. Фаза на Domain-Shader. Преземено на 2024-06-06.
43. „Geometry Shader Stage“. Direct3D 11 графика. Microsoft Docs. Преземено на 2024-06-06.
44. „Stream-Output Stage“. Direct3D 11 графика. Microsoft Docs. Преземено на 2024-06-06.
45. „RenderStates“. DirectX 9.0 за управуван код. Microsoft Docs. Преземено на 2024-06-06.
46. „Rasterizer Stage“. Direct3D 11 графика. Microsoft Docs. Преземено на 2024-06-06.
47. „Pixel Shader Stage“. Direct3D 11 графика. Microsoft Docs. Преземено на 2024-06-06.
48. „Output-Merger Stage“. Direct3D 11 графика. Microsoft Docs. Преземено на 2024-06-06.
49. „Google Announces ANGLE – Almost Native Graphics Layer Engine – so OpenGL ES can run over DirectX 9 – khronos.org news “. Khronos.org. Преземено на 2024-06-06.
50. „ANGLE“. Skia. Преземено на 2024-06-06.
51. „Google's Angle brings OpenGL to Windows“. I-programmer.info. Преземено на 2024-06-06.
52. „Chromium gets GPU acceleration – The H Open: News and Features“. H-online.com. Преземено на 2024-06-06.
53. Бриџ, Хенри (2010). „Chromium Blog: Introducing the ANGLE Project“. Blog.chromium.org. Преземено на 2024-06-06.
54. „angle/angle - Git at Google“. chromium.googlesource.com. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/ANGLE_(software)
55. ANGLE: OpenGL на Vulkan. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/ANGLE_(software)
56. „angleproject – ANGLE: Almost Native Graphics Layer Engine – Google Project Hosting “. Преземено на 2024-06-06.

57. „(WebGL) How to Enable Native OpenGL in your Browser (Windows)". Geeks3D. Преземено на 2024-06-06.
58. „Graphics on Windows from a different angle | Qt Blog". Qt. Преземено на 2024-06-06.
59. „ANGLE: Running OpenGL ES 2.0 Graphics Code on Windows". channel9.msdn.com. Преземено на 2024-06-06.
60. „WebGL 2.0 Achieves Pervasive Support from all Major Web Browsers ". Групата Хронос. Преземено на 2024-06-06.
61. „Cairo homepage". Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Cairo_(graphics)
62. „Skia Graphics Library". skia.org. Преземено на 2024-06-06.
63. Малик, Ом (2008). „Google Open Sources Skia Graphics Engine". gigaom. Преземено на 2024-06-06.
64. „Specialized Builds". Skia. Преземено на 2024-06-06. - https://en.wikipedia.org/wiki/Skia_Graphics_Engine
65. „The Skia 2D Graphics Library From Google - Phoronix". www.phoronix.com. Преземено на 2024-06-06.
66. „Skia Documentation". skia.org. Преземено на 2024-06-06.
67. https://en.wikipedia.org/wiki/Software_framework
68. https://en.wikipedia.org/wiki/Application_framework
69. https://en.wikipedia.org/wiki/Computer_graphics
70. https://en.wikipedia.org/wiki/Graphical_user_interface
71. https://en.wikipedia.org/wiki/Integration_tree
72. https://en.wikipedia.org/wiki/User_interface_design
73. https://en.wikipedia.org/wiki/Linear_interpolation
74. https://en.wikipedia.org/wiki/Graphics_library
75. https://en.wikipedia.org/wiki/List_of_3D_graphics_libraries
76. https://en.wikipedia.org/wiki/OpenGL
77. https://en.wikipedia.org/wiki/Vulkan
78. https://en.wikipedia.org/wiki/Metal_(API)
79. https://en.wikipedia.org/wiki/Graphics_Device_Interface
80. https://en.wikipedia.org/wiki/Direct3D

81. https://en.wikipedia.org/wiki/ANGLE_(software)
82. https://en.wikipedia.org/wiki/Skia_Graphics_Engine
83. https://en.wikipedia.org/wiki/Shader
84. https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/Materials/Editor/Interface/
85. https://en.wikipedia.org/wiki/Cairo_(graphics)
86. https://www.chromium.org/developers/design-documents/graphics-and-skia/
87. https://www.chromium.org/developers/design-documents/chromeviews/
88. https://skia.org/docs/user/color/
89. https://skia.org/docs/user/api/
90. https://skia.org/docs/user/sksl/