Математичка Гимназија

МАТУРСКИ РАД

из предмета *Рачунарство и информатика*

3D ГРАФИКА

Ученик
Марко Станојевић, IVe

Ментори
Снежана Јелић
Андреј Ивашковић

Београд, мај 2017.

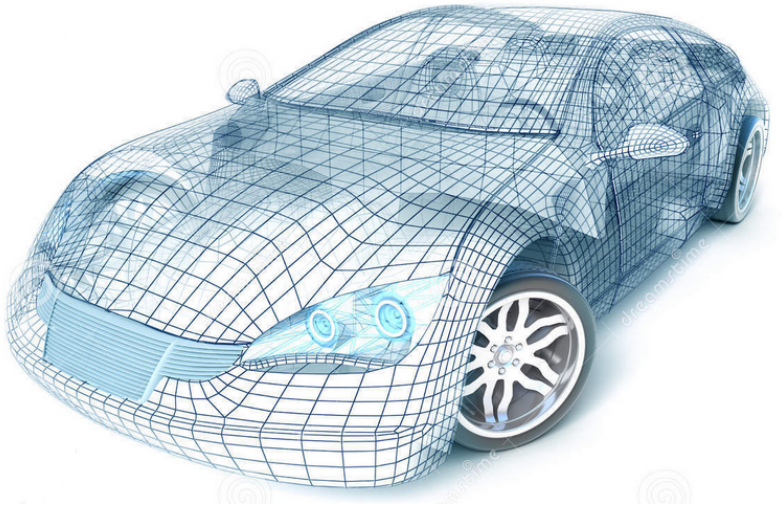
Садржај

1. Увод

1.1. Укратко о 3D графици

Потреба за приказивањем резултата рада рачунара постоји од када постоје и рачунари. Проблем приказивања резултата на екрану одувек је представљао и хардверски и софтверски изазов. Првобитни рачунари могли су да прикажу само текст на екранима. Напредовањем науке и технологије производње интегрисаних кола, постало је могуће на екранима контролисати и појединачне тачке - пикселе, што је означило почетак ере дводимензионалне, односно 2D графике. Убрзо се указала потреба за приказивањем слика тродимензионалних објеката на екранима рачунара, па је, најпре софтверски а потом и хардверски услед даљих унапређења и развоја технике, дошло до пробоја тродимензионалне графике у свет рачунара. Може се рећи да је 3D графика и даље једна од врло интензивно истраживаних области у савременом рачунарству.

Данас се 3D графика толико често користи да је постала уобичајена, а самим тим готово неприметна. Када у биоскопу погледамо најновији акциони филм увек најпре помислимо на заплет, а ретко кад (или никад) на софтвер који је био коришћен да би се тај филм произвео у великим студијима, као ни на хардверске системе који су били неопходни да би се то обавило за довољно кратко време. Данас се на рачунарима, конзолама и мобилним телефонима играју искључиво 3D игрице, без размишљања о графичким картицама и тимовима програмера који су заслужни за коначни изглед. Данас је права реткост видети раније уобичајен 2D платформер, *point-and-click* игрицу или било коју другу која не користи 3D графику – овакве игре данас постоје једино у домену *indie* софтверских девелопера. У ТВ рекламама гледамо 3D исцртане објекте, од пасти за зубе до најмодернијих аутомобила. Када седнемо у аутомобил, или видимо било који други индустријски произведени предмет или уређај, ни не помишљамо да се за њихову производњу користила 3D графика у студијима дизајнера који су тај производ могли разгледати из свих углова и пре него што је био произведен, као и машинских инжењера који су помоћу 3D графике направили машине, алате и калупе неопходне да се тај предмет произведе.



Слика 1: 3D жичани модел аутомобила

У развој 3D графике улаже се много новца и ентузијазма. Највећи улагачи су комерцијалне компаније, почев од оних који производе графичке картице и њихове процесоре, преко оних који развијају API-је као универзалне интерфејсе за програмирање тих картица, до оних које развијају софтверске апликације за њихову употребу у најразличитије сврхе. Поред ових компанија које из развоја 3D графике извлаче профит, на сцени су и тимови независних програмера отвореног кода (*open source*), који се првенствено баве развојем софтвера за 3D графику (API-ја и апликација) и који резултате свога рада чине доступним широј јавности путем интернета.

Укратко, 3D графика је постала значајан део садашњице, а њен развој су обележили многи изазови.

1.2. Развој 3D графике

Као и све друге рачунарске технологије, и 3D графика свој развој дугује постојању преплитања развоја хардвера и софтвера. Наиме, убрзо након што се створе нове хардверске могућности (попут бољих монитора и бржих процесора), појаве се и софтверске примене тих могућности у најразличитије сврхе. Брзим развојем софтвера ускоро се наиђе на ограничења постојећег хардвера, па се изнова створи потреба за његовим даљим развојем. И тако у круг.

Као што је уобичајено за све технологије везане за рачунаре, и рачунарска графика је за изузетно кратко време прешла дуг пут развоја. Цела армија софтверских инжењера, програмера и ентузијаста је током тог кратког раздобља радила на унапређењу рачунарске графике.

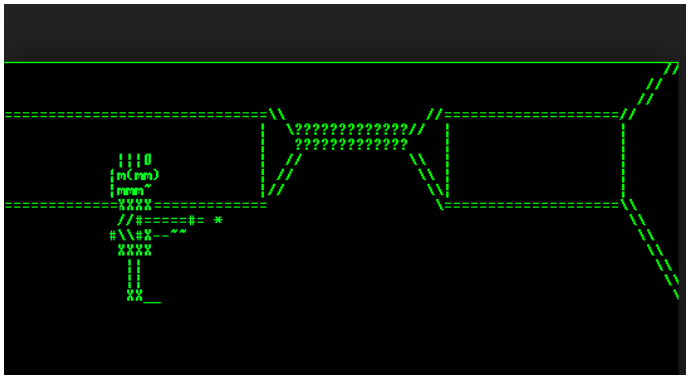
1.2.1. Текстуални приказ – матрица карактера и видео-генератори

3D графика је била незамислива у данима настанка рачунара. У то време није постојао ни 2D графички приказ, који би се састојао од тачкица (пиксела), већ се комплетан рад на рачунару сводио на унос и **текстуални приказ** карактера – слова, цифара и знакова.

Први рачунари (*mainframe*) били су изузетно гломазни и скупи за поседовање и одржавање, и могли су да их приуште само велики истраживачки центри и институције. Нису имали ни тастатуре ни мониторе, већ се унос инструкција и података обављао путем папирне траке или картица са рупама, док су се резултати штампали на штампачу (*teleprinter*).

Убрзо су штампачи тих великих рачунара били замењени конзолама сачињеним од тастатуре и екрана са црно-белом катодним цеви и видео генератором који је карактере претварао у видео сигнал. Програмери су се на разне начине довијали да и на таквим екранима остваре графички приказ, најчешће распоређивањем цртица, плусева и других симбола на екрану (*ASCII art*).

Услед даљег развоја технологије производње чипова јавила се могућност да се тржишту понуди приступачан рачунар за личну употребу у пословне сврхе. Интел (велики произвођач чипова) и IBM (велики произвођач пословних машина) произвели су PC 5150. То је био један од првих рачунара за пословну примену који није био конципиран по *mainframe-terminal* принципу, већ је био намењен за једног пословног корисника, па је добио назив персонални рачунар



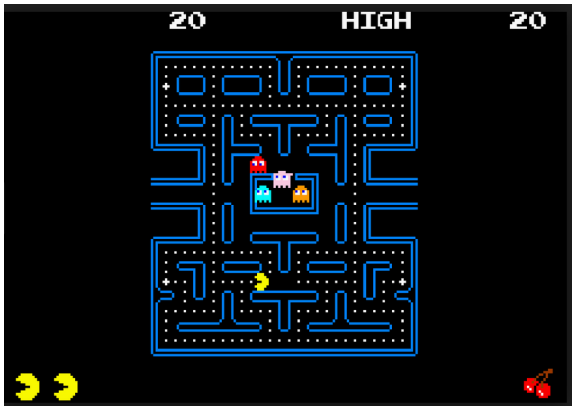
Слика 2: *ASCII art* игрица *DOOM* на конзоли VAX/VMS *mainframe*-а

(*personal computer* или скраћено PC). PC рачунари су имали наменску MDA (*Monohrome Display Adapter*) видео картицу са излазима за повезивање на монохроматски монитор који се добијао са рачунаром. Та видео картица је имала специјалну видео-меморију којој су истовремено могли да приступају и CPU и видео-генератор на видео картици. У ту видео-меморију би CPU уписао **матрицу карактера** за приказивање на монитору (најчешће 80×25), а истовремено би је **видео-генератор** читао и формирао видео сигнал за слање монитору. Поред тога, видео-генератор је био задужен и за приказ трепћућег курзора на задатој позицији на екрану.

1.2.2. 2D графика – матрица пиксела и 2D графички процесори

Услед повећаног интереса јавности за програмабилне калкулаторе и првобитне PC рачунаре, јавила се потреба да се тржишту понуди ценовно још приступачнији рачунар за личну употребу. Тада се појављују мали кућни рачунари попут ZX81, а нешто касније и ZX Spectrum, Commodore, Amiga и слични, који су имали TB модулаторе и видео излазе преко којих су могли да се повежу на антенски улаз кућног TB-а и на њему прикажу слику. Неки од рачунара тог времена попут Amstrad или Macintosh имали су мониторе дијагонале петнаестак цениметара. Притом, та слика није више била ограничена на приказ црно-белих карактера, већ је имала и могућност приказа појединачних пиксела различитих нивоа осветљености и боје. Те могућности, приступачна цена и интересовање јавности широм је отворило врата развоју софтверској индустрије, понајвише оне за развој игрица.

Истовремено, појавили су се PC XT, PC AT као и други чланови x86 династије која се и дан данас успешно развија. У њих су могле бити уграђене и колор графичке картице са излазима за колор мониторе, најпре по CGA (*Color Graphics Adapter*) а касније и по EGA (*Enhanced Graphics Adapter*), VGA (*Video Graphics Array*) и новијим стандардима. Те видео картице су поседовале већу количину видео-меморије у коју су, након постављања у жељени режим приказа, уместо карактера могле да сместе **матрицу пиксела** (320×200, 640×480 итд.). При томе за сваки пиксел био је намењен по један или више бајтова, у циљу приказивања различитих нивоа осветљености а касније и различитих боја. Те могућности су изнедриле потпуно другачији приступ за визуализацију резултата, чиме је почела ера интензивне употребе **2D графике**.

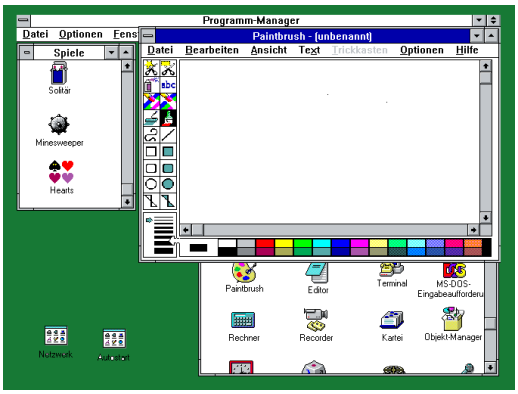


Слика 3: 2D графика у игрици *Pacman*

Иако се приказ састојао од појединачних пиксела, на њему се нису могле директно цртати графичке примитиве – линије, правоугаоници, кружнице итд. При томе, директно писање по видео меморији било је јако компликовано за програмере, првенствено због тога што су програмери морали самостално да пишу програме за исцртавање чак и основних графичких примитива попут линија и правоугаоника, пиксел по пиксел. Такав софтверски приступ исцртавању графике био је јако спор, јер се одвијао кроз магистралу рачунара.

Произвођачи видео картица су зато на графичку картицу додали **2D графички процесор** као наменски чип за попуњавање видео-меморије основним геометријским облицима: правоугаоницима, линијама и сл. Замисао је била да програмер, уместо заморног попуњавања пиксела у видео-меморији у циљу исцртавања неког графичког примитива, једноставно зада графичком процесору команду шта да исцрта, а да потом графички процесор тај посао обави неупоредиво брже, директним приступом видео-меморији.

Сваки произвођач видео картица је направио сопствени скуп команди за своје картице. Команде су биле међусобно некомпатибилне, а ни све картице нису умеле да исцртају све графичке примитиве. Иако су се графички програми коришћењем наменског графичког процесора значајно брже извршавали, посао програмерима није био нимало олакшан, јер су најпре морали да провере које су могућности графичке картице, те да у случају неподржавања неке од примитива и даље користе сопствени код за њено исцртавање пиксел по пиксел.



Слика 4: Графички оријентисан оперативни систем Windows 3.11

У то време почели су да се појављују првобитни графички оријентисани оперативни системи (OS). Они су решили проблем компатибилности графичких картица тиме што су на себе преузели бригу о томе која графичка картица се налази у рачунару и какве су јој хардверске могућности. Оперативни системи су створили софтверски интерфејс између хардвера графичке картице и програмера. То су постигли израдом драјвера и API-ја као скупа функција чијом употребом програмер задаје послове графичком процесору на видео картици. Свака API функција води рачуна да уколико исцртавање тражене графичке примитиве није директно подржано од стране графичког процесора, ипак исцрта ту примитиву софтверским путем – уписом у видео-меморију. Ова могућност графичких OS-ова дала је ветар у леђа софтверској индустрији, и убрзо је настало небројено много графичких програма за најразличитије примене.

1.2.3. 3D графика – троуглови у простору и 3D графички процесори

Није прошло много времена, а креативни програмери су почели да користе 2D графику за визуализацију објеката из 3D света приказивањем њихове пројекције на 2D мониторима, чиме су код посматрача стварали илузију 3D простора. Неколико графичких примена се потребом за 3D приказом издвајало од осталих: пословне графичке апликације (за пројектовање, дизајнирање и анимацију) као и програми за играње (игрице). Обе ове примене су значајно утицале на развој 3D графике, јер су као базу имале велики број корисника дубоких џепова.

Слика 5: 3D графика у игрици Doom

Најпре су посао приказа **3D графике** већим делом обављале саме апликације извршавањем на CPU-у. Прво би створиле дигитални модел 3D света у меморији рачунара, одређивањем x, y и z координата темена мноштва **троуглова у простору** који представљају "свет" у коме се налази виртуелни посматрач. Затим би апликација, извршена на CPU, искористила адекватне геометријске трансформације како би формирала 2D слику која представља пројекцију тих троуглова на видно поље посматрача. Потом би се та пројекција исцртавала у видео-меморији коришћењем API-ја. С обзиром на то да се највећи део тог посла обављао у CPU-у, било је потребно много програмерске умешности и оптимизација да би се добио довољно брз 3D приказ у циљу стварања анимације. И поред тога, као и чињенице да су тадашњи монитори имали ниску резолуцију, *frame-rate* таквих апликација био је низак, често на граници употребљивости.

Произвођачи графичких картица су тада направили **3D графичке процесоре** (3D GPU) који су способни самостално да обаве највећи део посла око 3D исцртавања. Проширили су количину меморије на видео картици и омогућили да се у њу поред пикселске мапе која се тренутно приказује на екрану може сместити и пикселска мапа која ће тек у следећем тренутку бити приказана на екрану (*double buffering*). Такође су омогућили да се у исту меморију смести и комплетан модел 3D света у виду скупа троуглова, битмапа односно текстура за "пресвлачење" тих троуглова, положаја и врсте светала, координата и оријентације камере и другог. С обзиром да се процес исцртавања троуглова из 3D света у *double buffered* пикселску мапу у великој мери могао паралелизовати, произвођачи су 3D GPU проширили великим бројем јединица за паралелно процесирање у циљу убрзања рада, што је било кључно за успех ове врсте графике. Такође, додали су нове команде којима је програмер графичкој картици могао саопштити од којих троуглова је сачињен 3D свет и друге параметре. Тиме су програмеру поједноставили поступак израде 3D графичке апликације, јер му је преостало само да створи дигитални модел 3D света и да га саопшти картици – о свему другом (то јест о приказу тог света на монитору) брине се сама картица, и то ради јако брзо. Програмер је стога ослобођен размишљања о томе **како** да нешто прикаже – већ само **шта** треба да прикаже у датом тренутку.

С обзиром на то да су произвођачи 3D GPU-ова (од којих су данас најпознатији NVIDIA и AMD) усвојили различите сетове 3D команди за њихове графичке картице, поново се указала потреба за стандардизацијом у циљу поједностављења писања програма за различите платформе. Стога је, као надградња нових хардверских могућности 3D GPU-ова графичких картица, настао **3D *engine***, као слој између 3D графичких картица и програмера.

Слика 6: Савремена 3D графика у игрици Elite Dangerous

Најпознатији 3D *engine* данас је DirectX који развија Мајкрософт и бесплатно дистрибуира у Windows оперативним системима, а поред њега треба поменути и OpenGL *engine* који се првенствено користи у пословној 3D графици (нпр. у комерцијалним програмима за моделирање). И други произвођачи, најчешће они који су своје искуство стекли израдом 3D апликација и игрица на различитим платформама (не обавезно Windows већ и на конзолама, Linux-у, Android-у и сл.), одлучили су да сопствене 3D *engine* понуде и другима, па тако данас постоје Frostbite, Unreal Engine, CryEngine и други. Неки 3D *engine*-и поједностављују и прорачун физике (статике и динамике кретања) 3D објеката и детекције њихових колизија.

1.3. Матурски рад и пројекат

Већина људи данас на питање шта је 3D графика интуитивно зна одговор. Међутим, на питање како рачунари приказују тродимензионалне облике на дводимензионалној површини екрана, ретко ко зна тачан одговор. Имајући то у виду, овај матурски рад има за циљ да покаже да је 3D рачунарска графика занимљива тема и да основни принципи нису тешки за разумевање као што се на први поглед чини.

У матурском раду детаљно сам описао комплетан поступак који се користи при формирању 3D графичког приказа, почев од моделовања објеката па све до приказивања слике. Такође, изложио сам математичку основу свих примењених метода. Да бих показао и практичну примену овог поступка, матурски рад сам засновао на софтверском пројекту који у потпуности прати изложени поступак. У пројекту сам направио графичку библиотеку Graphics3D, са циљем да се може употребити и при изради других пројеката. На крају сам направио и апликативни слој MGL којим сам демонстрирао употребу функција из библиотеке Graphics3D.

Да бих на што јаснији начин приказао комплетан поступак за приказивање 3D графике, одлучио сам да не користим хардверске 3D могућности савремених графичких картица, тако да сам целокупан прорачун почев од модела света до његовог приказа на екрану обавио софтверски. Поред функција за софтверско приказивање 3D графике, у библиотеку Graphics3D додао сам и потребне функције за моделовање објеката и извођење трансформација над њима.

2. Од модела и сцене до слике

У овом поглављу описујем поступак који је потребно обавити у циљу решавања основног проблема приказивања слике 3D света на екрану рачунара. Поступак се одвија у виду *pipeline*-а који се састоји од неколико фаза:

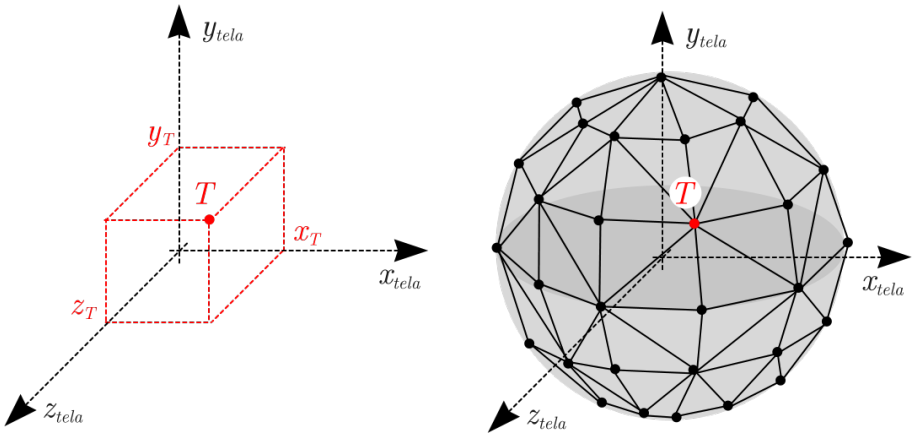
- Сваки појединачни објекат моделује се представљањем његових тачака[1] које формирају полигоне (најчешће троуглове) у циљу прекривања површине објекта;
- Сви објекти који се налазе у 3D свету смештају се на сцену линеарним математичким трансформацијама, које се могу представити матрично;
- Сцена (са свим објектима од којих је сачињена) пресликава се у простор посматрача, што се такође обавља матричним трансформацијама;
- Одбацује се део сцене коју посматрач не може видети, било зато што се налази иза посматрача, ван његовог видног поља, преблизо или предалеко од њега;
- Преостали део сцене се пројектује на замишљени прозор кроз који посматрач гледа. Том приликом води се рачуна о перспективи, међусобном заклањању објеката и њихових делова, промени боје објекта услед различитог осветљења и слично;
- На крају се слика са замишљеног прозора приказује на монитору рачунара.

2.1. Сопствени модел објекта

Да би се 3D свет могао приказати на екрану, најпре га је потребно математички **моделовати**. Са обзиром на то да је простор веома комплексан, није могуће извести моделовање света до свих његових најситнијих детаља. Зато се прибегава прављењу поједностављених модела објеката који се у том свету налазе – који ће бити довољно детаљни да из угла посматрача изгледају реално, а истовремено довољно упрошћени да се математички могу брзо обрадити.

За потребе моделовања објекта може се изабрати коначан број значајних тачака са његове површине које га довољно добро представљају. За сваку од тих тачака чувамо њене x, y и z координате у координатном систему објекта (у односу на произвољни координатни почетак који је дизајнер изабрао). Интуитивно је рећи да што је већи број тих значајних тачака, то ће оне боље представљати посматрани објекат. Те тачке се међусобно могу повезати дужима, а које опет формирају троуглове који леже на површини објекта. Овај процес се завршава онда када се површина објекта потпуно покрије довољно ситним троугловима – теселише. Коначно, овакав модел тела састоји се од облака тачака на површини тела, односно од **скупа троуглова** који их спајају (*mesh*).

[1] Под тачком подразумевамо њен вектор положаја (радијус-вектор) у сопственом 3D координатном систему.

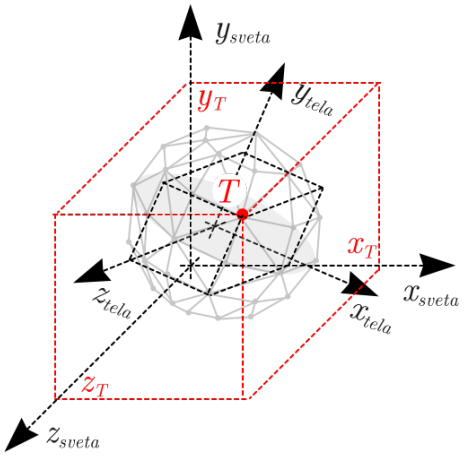


Слика 7: Представљање површине лопте скупом троуглова

Наравно, постоје и другачији површински модели, код којих се објекти представљају скупом полигона или кривих, као и запремински модели, али они у овом раду нису анализирани.

2.2. Постављање објеката на сцену

Пошто су координате тачака на површини тела математички описане у његовом сопственом координатном систему, да би се тело сместило на сцену на којој се налазе и остала тела тог света, потребно је извршити трансформацију тачака тела из сопственог у светски координатни систем. Трансформација се математички обавља множењем свих тачака тела **трансформационом матрицом**. У зависности од тога како је трансформациона матрица израчуната, могу се извршити различите трансформације тела; на пример транслаторно кретање у простору, скалирање, ротације, ... При томе, уколико је потребно извршити неколико узастопних трансформација на објекту, и даље је довољна једна трансформациона матрица, која је израчуната као производ матрица за сваку појединачну трансформацију, у жељеном редоследу извршавања.



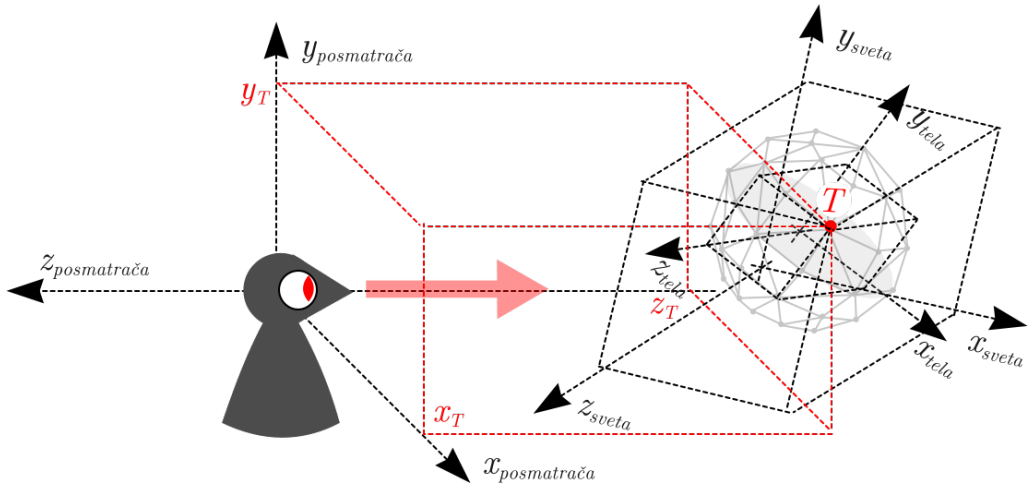
Слика 8: Трансформација тачака тела у светски координатни систем

Уколико је неки објекат на сцени чврсто повезан за други објекат, најпре га је потребно трансформисати у координатни систем тог другог објекта, а

потом и над њим применити исту трансформацију која се примењује над тим другим објектом. То је могуће учинити и рекурзивно (над више хијерархијски зависних објеката). Јасно је да је за целокупну операцију пресликавања тачака сваког од тих објеката у светски координатни систем и даље довољна једна трансформациона матрица.

2.3. Постављање посматрача на сцену

Иако се и посматрач налази на сцени, уместо трансформације његових координата у светски координатни систем врши се обрнута трансформација – све координате објеката на сцени које су дате у светском координатном систему се трансформишу у координатни систем посматрача.



Слика 9: Трансформација у координатни систем посматрача

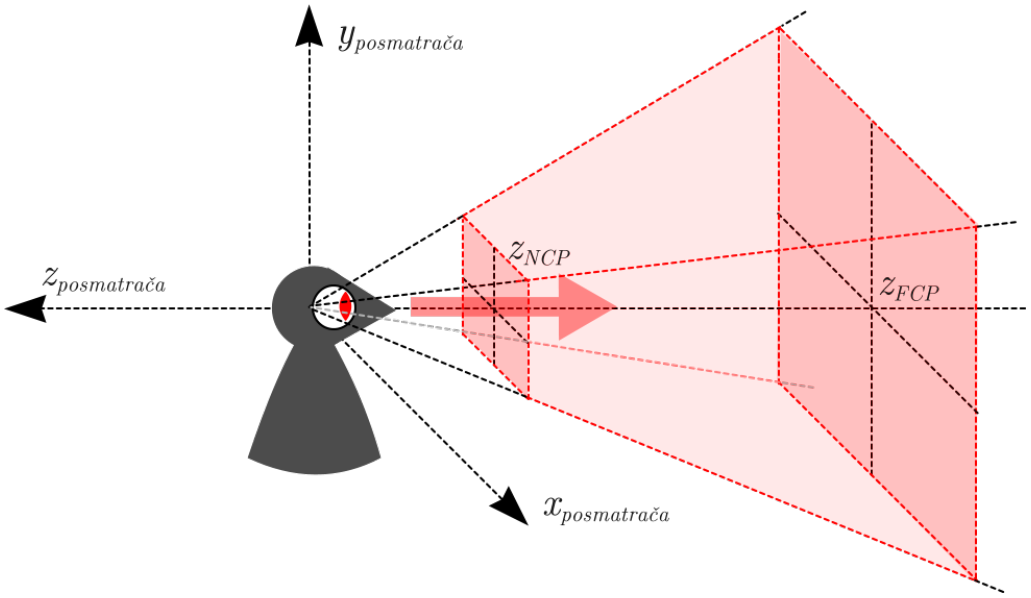
Трансформација у координатни систем посматрача се математички такође обавља множењем тачака тих објеката **трансформационом матрицом посматрача** што значи да је могуће извести померање и ротацију посматрача (односно смера у коме гледа) као и друге операције. При томе, сматра се да се посматрач налази у центру свог координатног система, и да му је поглед усмерен у негативном смеру z-осе.

Треба водити рачуна да се трансформациона матрица посматрача рачуна као слика у огледалу у односу на трансформационе матрице објеката. На пример, уколико се посматрач налази удаљен $+200$ јединица по z-оси у светском координатном систему, то значи да трансформациону матрицу посматрача треба израчунати тако да све објекте на сцени помери за -200 јединица по z-оси. Или, на пример, уколико посматрач ротира своје видно поље улево, све објекте на сцени треба ротирати удесно у односу на посматрача.

2.4. Део сцене коју посматрач види

Поглед на свет из угла посматрача је ограничен реалним могућностима његовог ока. Око је у основи сачињено од очног сочива које се налази испред очног дна - мрежњаче. Слика коју посматрач види формира се проласком

светлосних зрака кроз очно сочиво, и пројектовањем те слике на мрежњачу на којој се налазе фото-осетљиве ћелије. Око има ограничено видно поље, односно не може да види целокупних 360° околине, већ по хороизонтали око 160° а по вертикали 135°.



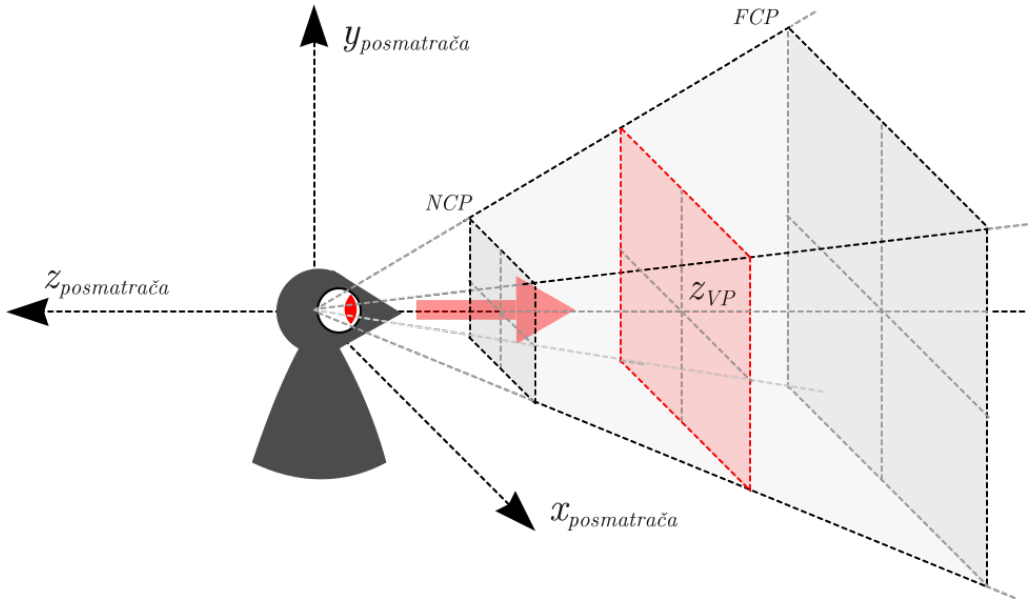
Слика 10: Део сцене коју посматрач види је у облику зарубљене пирамиде

Простор који око види подсећа на купу чији врх се налази у жижи у самом оку посматрача, а дно далеко испред посматрача, у смеру у коме посматрач гледа. Услед постојања очног сочива, око мутно види тела која се налазе превише близу, а у зони између сочива и очног дна се тела и не могу наћи. Такође, услед постојања коначног броја ћелија на очном дну, око уопште не види тела која су превише далеко, уколико њихова пројекција на очно дно постане ситнија од разлучивости ока. Из истог разлога око не види ни тела која су превише мала, иако се налазе близу посматрача. Због тога се може рећи да посматрач види део света облика **зарубљене правилне пирамиде**, односно да види само оне објекте који се (у целини или само делимично) налазе унутар ње. Ради уштеде процесорског времена, остали објекти који се не могу видети се не процесирају.

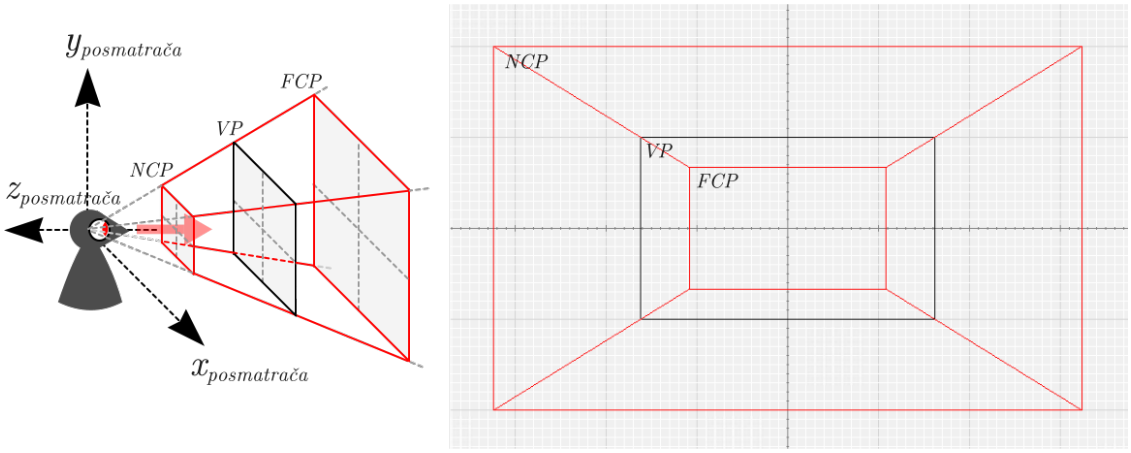
2.5. Пројектовање слике на прозор

Посматрач гледа објекте кроз правоугаоник посматрања – **прозор** (*viewport*). Прозор је одабрани пресек пирамиде посматрања, који је такође и паралелан базама пирамиде. На прозор се пројектује слика на начин како је посматрач види у виртуелном свету. Та слика ће у стварности бити приказана на монитору рачунара, чиме ће код реалног посматрача створити илузију да се заправо он налази на месту посматрача у моделованом 3D свету, те да у тај свет гледа кроз екран рачунара као кроз прозор.

Пројектовање на прозор се обавља уз поштовање правила перспективе. Ова трансформација се обавља множењем координата тачака објеката **пројекционом матрицом**, која увећава тела која се налазе између посматрача и прозора



Слика 11: Прозор на који се пројектује слика коју посматрач види

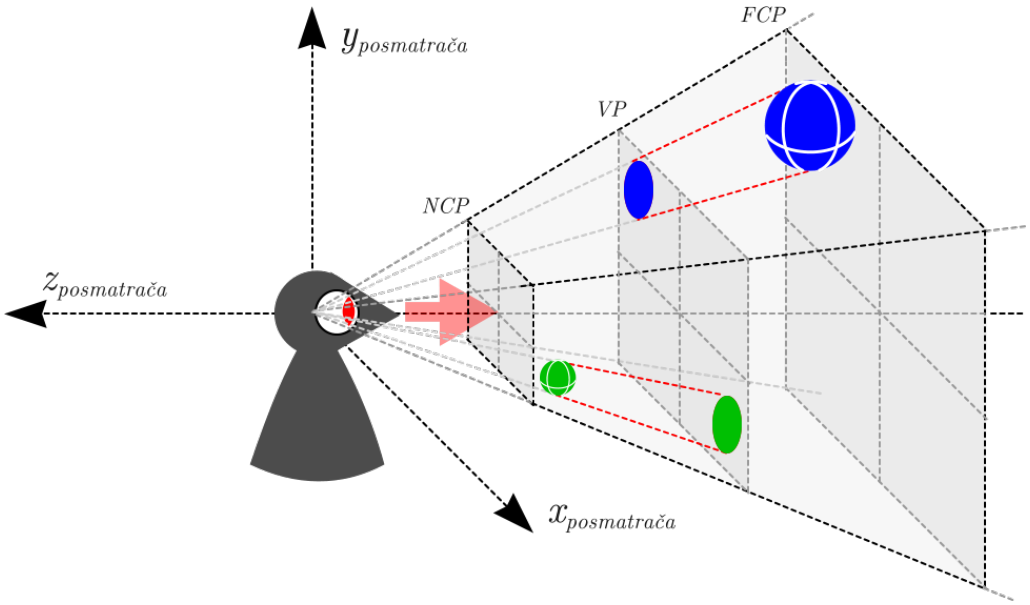


Слика 12: Пирамида посматрања и прозор виђени из угла посматрача

а смањује тела која се налазе иза прозора, пропорционално њиховој удаљености од прозора.

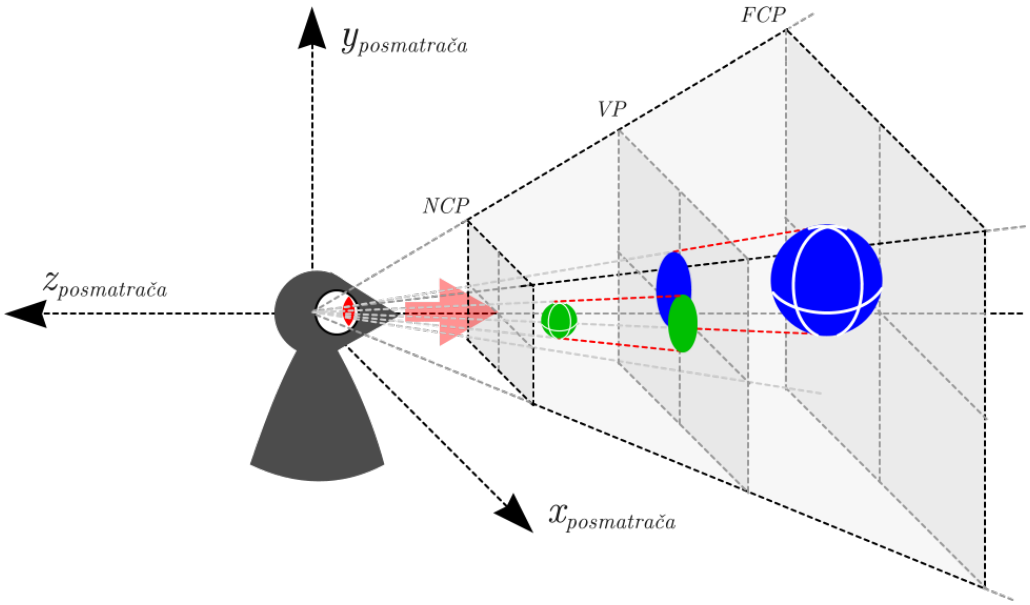
Приликом пројектовања на прозор, поред перспективе потребно је водити рачуна и о међусобном положају објеката које посматрач види, односно о **удаљености објеката од посматрача**. Неки од тих објеката су ближи посматрачу од других и, зависно од релативног положаја, могу делимично или у целини заклонити поглед посматрача на даље објекте (у овом раду сам, ради једноставности, подразумевао да су сва тела непрозирна).

Да би се слика пројектовала онако како је посматрач види, неопходно је водити рачуна о видљивости објеката по дубини, односно не исцртавати она (даља) тела (или њихове делове) која су заклоњена другим (ближим) телима, гледано из угла посматрача. Такође, свако тело има своју предњу (ближу) и задњу (даљу) страну гледано из угла посматрача, па је потребно водити рачуна да се троуглови који представљају задњу страну тела не приказују на слици. На крају, услед могућег запреминског преклапања објеката због кога није могуће



Слика 13: Увећање блиских и смањивање удаљених објеката при пројектовању на прозор

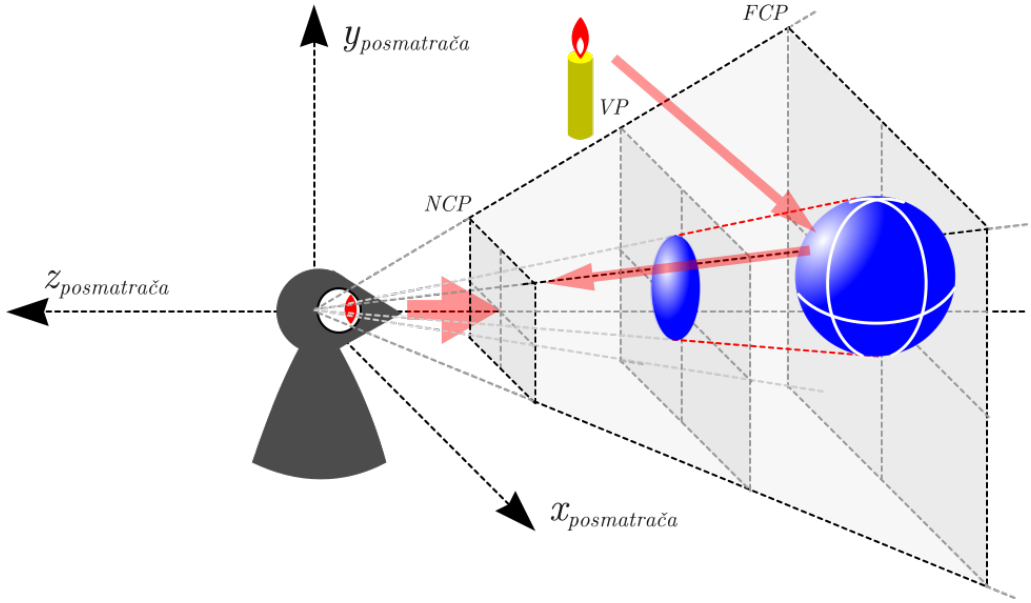
установити који објекат је ближи посматрачу а који даљи, потребно је водити рачуна да се правилно прикажу и међусобно пресечени троуглови тих објеката. У сврху решавања овог проблема, у пројекту који прати овај рад применио сам два приступа: *back-front* алгоритам и *Z-buffer* алгоритам.



Слика 14: Преклапање пројекција тела

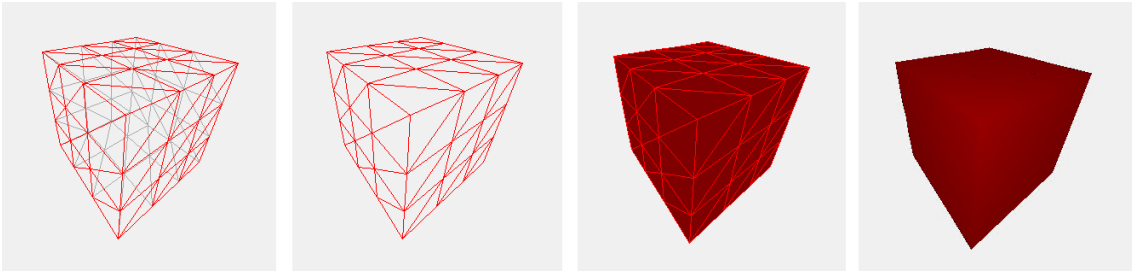
Објекти могу бити обојени различитим бојама (ради поједностављења модела у овом раду се нисам бавио текстурама). У зависности од положаја извора светла, те боје посматрачу могу изгледати светлије или тамније. При пројектовању на прозор, неопходно је одредити угао под којим се светлосни зрак из извора светла одбија од површине објекта, и на основу њене дифузности одредити боју којом треба исцртати тај троугао односно сваки појединачни пиксел

унутар троугла. Овај поступак се зове **сенчење** (*shading*). У пројекту који прати овај рад имплементирао сам тачкасто осветљење које се налази у посматрачевом оку, а применио сам *fixed-colour shading* као и *flat shading* алгоритам.



Слика 15: Тачкасти извор светла и сенчење

У овом раду бавио сам се **жичаним** (*wireframe*) и **површинским** (*surface*) моделима објеката, док **запреминski** (*solid*) модел нисам реализовао у циљу једноставности. Приликом исцртавања жичаног модела поступак је једноставнији него код површинског модела, јер се у потпуности прескаче оно што је наведено у претходна три пасуса.



Слика 16: Жичани (лево) и површински (десно) модели коцке

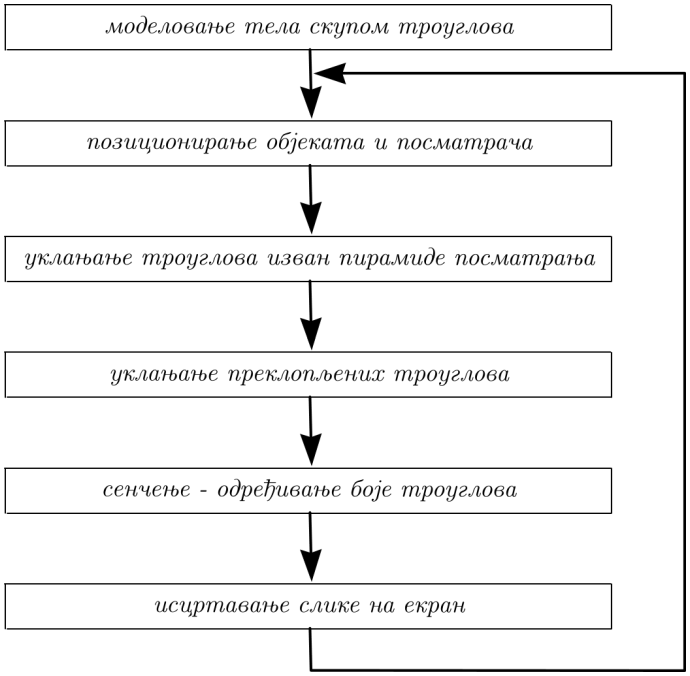
2.6. Фрејмови и анимација

До сада смо подразумевали да се поступак исцртавања 3D слике спроводи над непокретним моделом, "замрзнутим" у тренутку посматрања. На тај начин добијена слика представља само један **фрејм** (*frame*, kadar), односно слику коју треба приказати у тачно једном тренутку. Уколико се ради о моделу променљивом у простору, где се објекти и посматрач крећу по сцени, претходно приказани фрејм већ у следећем тренутку више не важи, и потребно је израчунати нови. Фрејмови који се у кратким и правилним временским тренуцима смењују на екрану посматрача стварају илузију кретања објеката.

Приликом израчунавања новог фрејма, потребно је:

1. променити моделе објеката (уколико долази до њихове деформације);
2. променити трансформационе матрице објеката (уколико се објекти крећу по сцени);
3. променити трансформациону матрицу посматрача (уколико се посматрач креће по сцени);
4. променити пројекциону матрицу (уколико долази до промене перспективе посматрања).

Након тих промена, треба поновити поступак за исцртавање нове слике. Понављањем ових операција у кратким временским тренуцима настаје анимација.



Слика 17: Анимација настаје понављањем наведених операција више пута у секунди

Међутим, том приликом се не сме дозволити да слика на екрану буде приказана недовршена. Зато се, уместо директног цртања у видео-меморију (а која се у непрекидно приказује на екрану), за цртање користи додатна меморија. Тек након што се цртање у њу комплетно заврши, онда се њене вредности уписују у видео-меморију, чиме се софтверски остварује *double buffering*.

Да би анимација текла глатко, број фрејмова приказаних у једној секунди мора бити најмање 25 (мада је пожељно бар двоструко више), јер темпорални режањ посматрача — услед инерције — не може да региструје брже промене. То значи да за рачунање и исцртавање једног фрејма рачунар на располагању има мање од 0.04 секунде. Због тога је, посебно у случају сложенијих модела представљених са много троуглова, неопходно вршити различите оптимизације

како се процесорско време не би трошило на прорачунавање троуглова који се на слици неће видети. Једна од оптимизација је прескакање анализе свих троуглова оног објекта, за који се унапред установи да га посматрач уопште не може видети. Друга врста оптимизације је коришћење више процесорских нити (*threads*) у циљу њиховог паралелног извршавања на више процесорских језгара. Наравно, друга оптимизација се значајно више користи у графичким картицама, због тога што графички процесори имају могућност паралелног израчунавања које се одлично примењује у овом контексту.

3. Координате и трансформационе матрице

3.1. 4D вектори

У претходној секцији дефинисали смо тело као облак тачака (*point cloud*) и скуп дужи (*vertices*) које те тачке спајају, и које тиме сачињавају троуглове. Међутим, није једнозначно одређено како треба математички моделовати тачку у простору. На први поглед тачке у 3D би се могле представити као тродимензионалан вектор:

$$\vec{u} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{1}$$

због тога што су описане са три координате – x, y, z. Међутим, да би се раније поменуте трансформације представиле на једноставнији начин, користе се 4D вектори у хомогеним координатама:

$$\vec{u} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \tag{2}$$

где $w \neq 0$. Овакав координатни систем — са једном додатном координатом — чини **хомогене координате**. Он умногоме олакшава рад са афиним трансформационим матрицама (о којима ће бити више речи у следећој секцији) због тога што оне морају бити 4D, а множење вектора матрицом мора бити димензионо компатибилно – 4×4 матрица не може множити 3D вектор.

Неке од особина хомогених координатних система су:

→ Често очигледнија симетрија формула у односу на базисни координатни систем; увек имају једну више димензију од базисног координатног система, и та додатна димензија најчешће води поједностављењу формула.

→ Вектори у хомогеном координатном систему се могу разбити на класе еквиваленције:

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \mapsto \begin{bmatrix} x/w \\ y/w \\ z/w \\ 1 \end{bmatrix} \tag{3}$$

→ 3D вектори се веома једноставно претварају у одговарајуће 4D векторе:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \mapsto \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4}$$

→ 4D вектори одговарају тачно једном 3D вектору:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{5}$$

→ Тачке у бесконачности се могу забележити као уређена четворка **коначних** бројева (ако дозволимо $w = 0$).

У матурском раду ћемо користити хомогени координатни систем димензије 4, јер је базисни координатни систем димензије 3. Такође, ради прегледности, вектори ће понегде бити писани као транспоновани (али ће се свеједно множити са матрицама са леве стране).

Да би смо конструисали векторски простор $\mathcal{V}$, потребно је дефинисати следеће

Дефиниција 1

Структура $\mathcal{V} = (\mathbb{V}, +, \cdot, \mathbb{K})$ је векторски простор ако важи следеће:

1) $(\mathcal{V}, +)$ је *Абелова* група, где је $\mathcal{V}$ непразан скуп вектора (носач);
2) $(\mathbb{K}, +, \cdot)$ је **поље** скалара;
3) операција $+ : \mathbb{V} \times \mathbb{V} \to \mathbb{V}$ је дефинисана
4) операција $+ : \mathbb{K} \times \mathbb{V} \to \mathbb{V}$ је дефинисана
5) И важе наведене аксиоме за свако $\alpha, \beta \in \mathbb{K}$ и $x, y \in \mathbb{V}$:
   - i $\alpha(x + y) = \alpha x + \alpha y$
   - ii $(\alpha + \beta)x = \alpha x + \beta x$
   - iii $\alpha(\beta x) = (\alpha\beta)x$
   - iv $1 \cdot x = x$.

Ми се у овом раду нећемо бавити показивањем да је $\mathcal{V}$ заиста векторски простор, већ ћемо само дефинисати за 3D графику потребне ствари:

1. Скуп вектора $\mathbb{V}$
   a. $\mathbb{V} = \mathbb{R}^3 \times (\mathbb{R} \setminus \{0\})$ облика
   $$\vec{u} = \begin{bmatrix} x & y & z & w \end{bmatrix}^T \tag{6}$$
   b. Укључујући нула вектор
   $$\vec{0} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T \tag{7}$$
2. Поље скалара $\mathbb{K}$ — у нашем случају $\mathbb{R}$

3. Операцију сабирања вектора

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ w_1 \end{bmatrix} + \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} x_1 w_2 + x_2 w_1 \\ y_1 w_2 + y_2 w_1 \\ z_1 w_2 + z_2 w_1 \\ w_1 w_2 \end{bmatrix} \tag{8}$$

4. Операцију множења вектора скаларом

$$k \cdot \begin{bmatrix} x & y & z & w \end{bmatrix}^T = \begin{bmatrix} x & y & z & w \end{bmatrix}^T \cdot k = \begin{bmatrix} kx & ky & kz & w \end{bmatrix}^T \tag{9}$$

Над истим векторским простором $\mathcal{V}$ су дефинисане и додатне операције:

1. Скаларно множење вектора

$$\begin{bmatrix} x_1 & y_1 & z_1 & w_1 \end{bmatrix}^T \cdot \begin{bmatrix} x_2 & y_2 & z_2 & w_2 \end{bmatrix}^T = (x_1 x_2 + y_1 y_2 + z_1 z_2)/(w_1 w_2) \tag{10}$$

2. Векторско множење вектора

$$\begin{bmatrix} x_1 & y_1 & z_1 & w_1 \end{bmatrix}^T \times \begin{bmatrix} x_2 & y_2 & z_2 & w_2 \end{bmatrix}^T = \begin{bmatrix} \begin{vmatrix} y_1 & z_1 \\ y_2 & z_2 \end{vmatrix} & \begin{vmatrix} z_1 & x_1 \\ z_2 & x_2 \end{vmatrix} & \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & w_1 w_2 \end{bmatrix}^T \tag{11}$$

које се често користе у 3D графици, приликом *clipping*-а и пројекције на екран.

3.2. Матрице значајних трансформација

Битно је нагласити да су све операције над векторима које се користе у 3D графици *афине* — трансформишу тела *хомогено* — осим перспективне трансформације, која се примењује на самом крају израчунавања и која деформише тела тако да заузимају већу површину што су ближе посматрачу.

Афине трансформације су следеће:

1. Ротација

   a. Око x-осе за угао t у математичком смеру

$$\mathbf{R}_x(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos t & -\sin t & 0 \\ 0 & \sin t & \cos t & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{12}$$

   b. Око y-осе за угао t у математичком смеру

$$\mathbf{R}_y(t) = \begin{bmatrix} \cos t & 0 & \sin t & 0 \\ 0 & 1 & 0 & 0 \\ -\sin t & 0 & \cos t & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13}$$

c. Око z-осе за угао t у математичком смеру

$$\mathbf{R}_z(t) = \begin{bmatrix} \cos t & -\sin t & 0 & 0 \\ \sin t & \cos t & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

d. Произвољна ротација (према конвенцији)

$$\mathbf{R}(t_x, t_y, t_z) = \mathbf{R}_z(t_z) \cdot \mathbf{R}_y(t_y) \cdot \mathbf{R}_x(t_x) \tag{15}$$

— редослед вршења ротација је следећи: $\mathbf{R}_x \to \mathbf{R}_y \to \mathbf{R}_z$

2. Скалирање за k_x, k_y, k_z по x, y, z, редом, у позитивном смеру

$$\mathbf{S}(k_x, k_y, k_z) = \begin{bmatrix} k_x & 0 & 0 & 0 \\ 0 & k_y & 0 & 0 \\ 0 & 0 & k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{16}$$

3. Транслација за l_x, l_y, l_z по x, y, z, редом, у позитивном смеру

$$\mathbf{T}(l_x, l_y, l_z) = \begin{bmatrix} 1 & 0 & 0 & l_x \\ 0 & 1 & 0 & l_y \\ 0 & 0 & 1 & l_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{17}$$

Значајне су и следеће матрице, иако нису *афине*:

1. Пројекција на раван паралелну XY-равни, и удаљена од ње за $+d$ у правцу z-осе

$$\mathbf{P}_{XY}(d) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \tag{18}$$

2. *Frustrum culling* матрица (имплементирана у OpenGL-у)

$$\mathbf{FC}(n, f, t, b, l, r) = \begin{bmatrix} 2n/(r-l) & 0 & (r+l)/(r-l) & 0 \\ 0 & 2n/(t-b) & (t+b)/(t-b) & 0 \\ 0 & 0 & -(f+n)/(f-n) & -2fn/(f-n) \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{19}$$

— ова матрица је **обједињена** матрица матрица пројекције и скалирања, те за нас није од нарочите важности.

3.3. Примене трансформација при промени координата

Све горенаведене трансформације се користе у 3D графици, али на различитим стадијумима исцртавања.

Пример 1. *Имамо вектор*

$$\vec{u} = \begin{bmatrix} 2 & 3 & 4 \end{bmatrix}^T$$

Рецимо да желимо да прво ротирамо вектор око x-осе за угао $\alpha = \pi/2$ (математички смер), да би га потом скалирали $5\times$ у односу на x-осу, и транслирали за 5 јединица у негативном смеру паралелно x-оси. Како то извести?

→ *Прво трансформишемо вектор у хомогене координате:*

$$\vec{u_0} = \begin{bmatrix} 2 & 3 & 4 & 1 \end{bmatrix}^T \tag{20}$$

→ *Затим га помножимо са леве стране са матрицом ротације:*

$$\vec{u_1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\pi/2) & -\sin(\pi/2) & 0 \\ 0 & \sin(\pi/2) & \cos(\pi/2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 4 \\ 1 \end{bmatrix} \tag{21}$$

→ *Потом га скалирамо:*

$$\vec{u_2} = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\pi/2) & -\sin(\pi/2) & 0 \\ 0 & \sin(\pi/2) & \cos(\pi/2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 4 \\ 1 \end{bmatrix} \tag{22}$$

→ *Након тога га транслирамо:*

$$\vec{u_3} = \begin{bmatrix} 1 & 0 & 0 & -5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\pi/2) & -\sin(\pi/2) & 0 \\ 0 & \sin(\pi/2) & \cos(\pi/2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 4 \\ 1 \end{bmatrix} \tag{23}$$

→ *Сада објединимо све матрице у једну помоћу матричног множења:*

$$\vec{u_3} = \begin{bmatrix} 5 & 0 & 0 & -5 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 4 \\ 1 \end{bmatrix} \tag{24}$$

→ *Потом помножимо вектор са обједињеном матрицом:*

$$\vec{u_3} = \begin{bmatrix} 5 & -4 & 3 & 1 \end{bmatrix}^T \tag{25}$$

→ *И коначно вратимо вектор из хомогеног у Декартов координатни систем:*

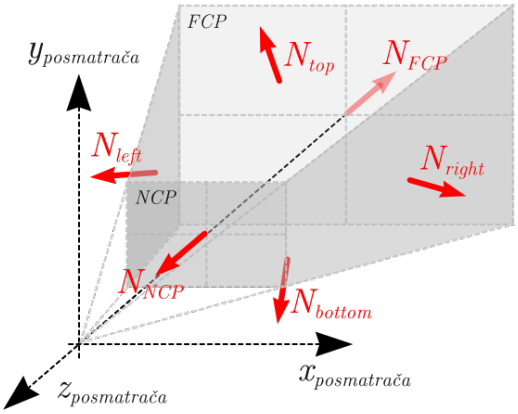
$$\vec{u_4} = \begin{bmatrix} 5 & -4 & 3 \end{bmatrix}^T \tag{26}$$

Овај пример илуструје како се трансформише један вектор у 3D простору у данашњим графичким процесорима. Међутим, тела се састоје из више троуглова, сваки од којих има три вектора темена, тако да се за трансформацију тела мора на свакој тачки темена свих троуглова на телу применити нека трансформациона матрица. Ако се ради о истој трансформационој матрици, тада кажемо да се тело понаша као чврсто.

4. Формирање слике

4.1. *Clipping*

Пре пројектовања троуглова на прозор, неопходно је изоставити оне троуглове који се не могу видети јер се у целини налазе изван пирамиде посматрања. У ту сврху се користи ***clipping*** – поступак одбацивања троуглова који се неће видети, као и замене троуглова који би се *делимично* видели другим троугловима који ће се у *целини* видети.



Слика 18: Нормале на пирамиду посматрања

Поступак *clipping*-а обавља се идентично над сваком појединачном равни пирамиде посматрања, којима се најпре доделе вектори нормала тако да су оријентисани ка спољашности пирамиде. Након тога за сваки троугао врши се испитивање да ли му се бар једно теме налази са једне и истовремено бар једно теме са друге стране *clipping* равни. Уколико се то деси, прорачунају се тачке пробоја дужи страница троугла кроз раван.

Тачка пресека дужи троугла и равни се рачуна по формули:

$$\vec{v_p} = \vec{v_1} + \frac{(\vec{L} - \vec{v_1}) \cdot \vec{N}}{(\vec{v_2} - \vec{v_1}) \cdot \vec{N}} \cdot (\vec{v_2} - \vec{v_1}) \tag{27}$$

где су:

$\vec{v_1}$ и $\vec{v_2}$ – вектори крајњих тачака дужи,

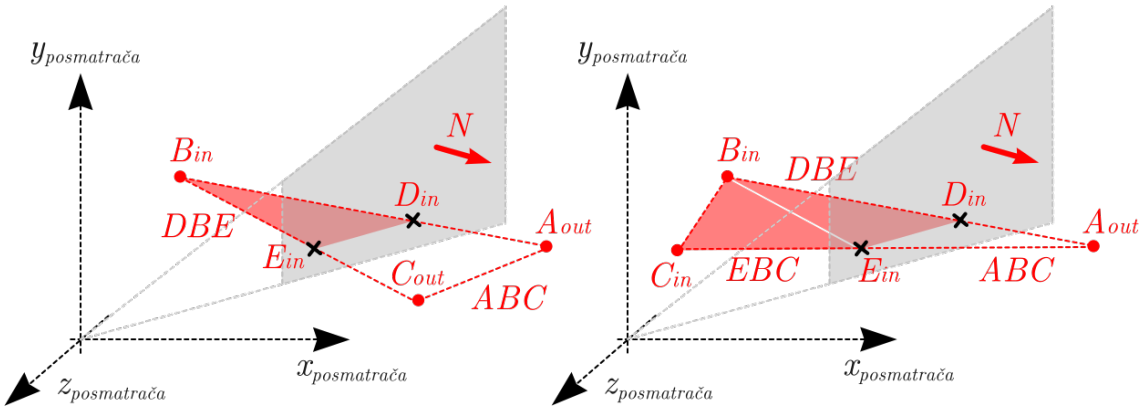
$\vec{L}$ – вектор произвољне тачке у равни,

$\vec{N}$ – вектор нормале на раван.

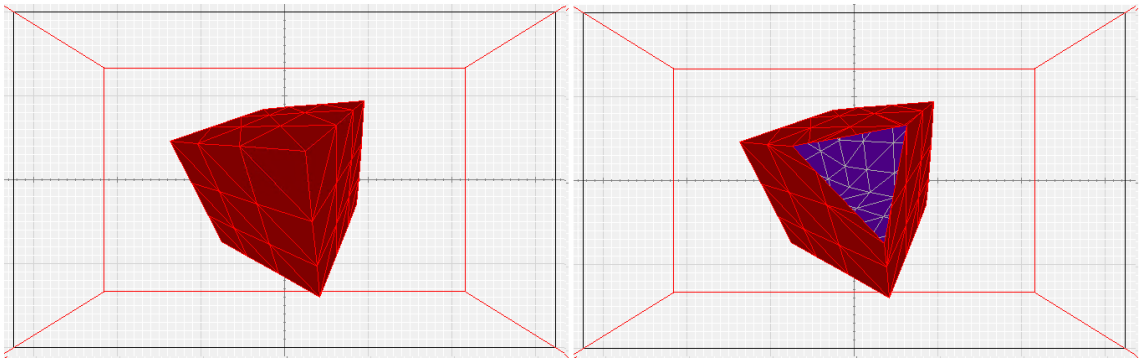
Након одређивања пробојних тачака, пресечени троугао замени са једним или са два друга мања троугла, који се налазе са унутрашње стране равни. При замени трогулова мањим, неопходно је водити рачуна да се сачува првобитна оријентација троугла, односно да и заменски троуглови буди једнако оријентисани у простору као и почетни троугао.

Уколико се сва три темена троугла налазе са спољне стране *clipping* равни, тај троугао се у целости изоставља из даљег процесирања. Поступак *clipping*-а

се завршава када преостану само троуглови који се у целости налазе са унутрашње стране свих *clipping* равни.

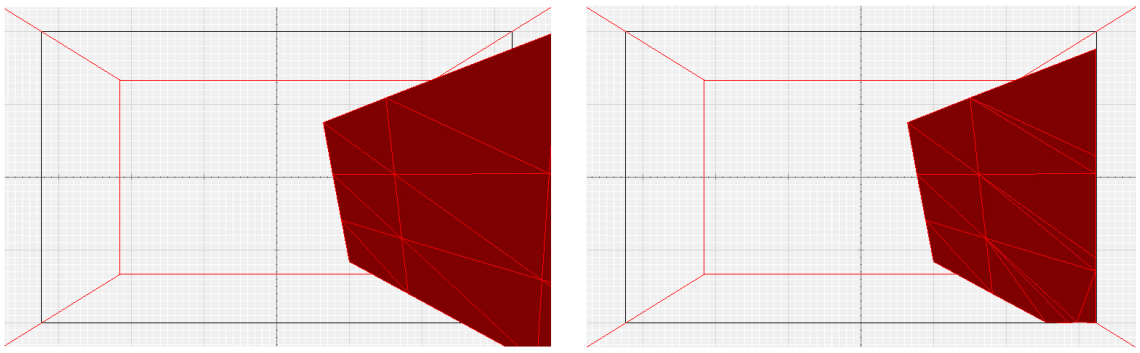


Слика 19: Замена троугла ABC мањим троугловима током *clipping*-а



Слика 20: *Clipping* коцке по NCP (десно – након *clipping*-а уочава се унутрашњост коцке)

Изостављање троуглова је посебно битно учинити за троуглове који се налазе јако близу ока посматрача, а поготово оне чија темена би се нашли у самом оку, односно у жижи посматрања. Таква темена не могу бити пројектована услед неодређености места на којем их треба пројектовати. Стога се поступак *clipping*-а мора обавезно обавити над NCP (*Near Clipping Plane*) равни која се налази на врху пирамиде посматрања.



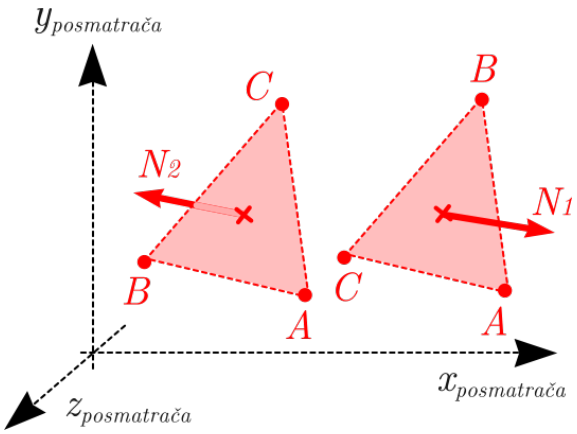
Слика 21: *Clipping* коцке по бочним странама пирамиде посматрања (десно – након *clipping*-а уочавају се додатни троуглови на површини)

У FCP (*Far Clipping Plane*) равни на дну пирамиде посматрања, као ни у бочним равнима пирамиде посматрања није обавезно применити поступак *clipping*-а. Међутим, *clipping* и на тим равнима је то ипак пожељно учинити, и то у што ранијој фази исцртавања, да би се уштедело процесорско време. Ово је изузетно значајно ако је брзина исцртавања битна, посебно када је реч о анимацијама.

4.2. *Back-front* алгоритам

Следећи корак при пројектовању троуглова на прозор је установљавање њиховог евентуалног преклапања. У ту сврху се могу користити различити алгоритми, са својим предостима и манама у односу на друге.

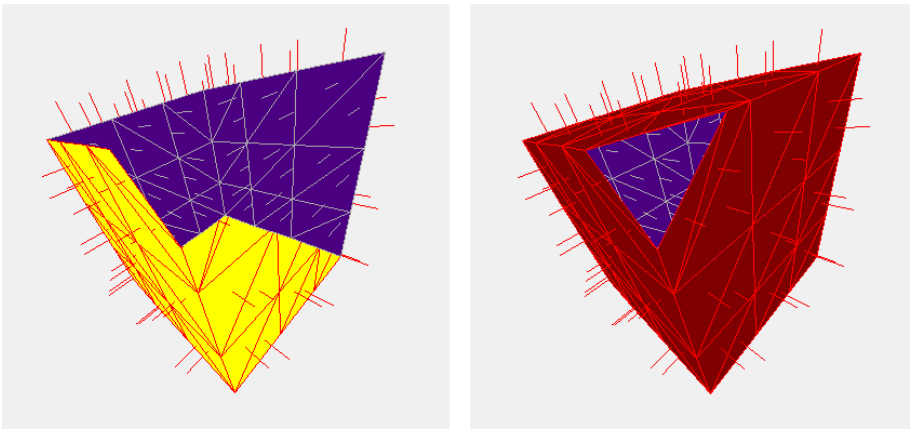
Један од једноставнијих алгоритама за ову сврху је ***back-front*** алгоритам. У њему се на елегантан начин заобилази провера преклапања, и краси га велика брзина јер се за исцртавање могу користити брзе функције 2D *engine*-а графичке картице. Нажалост, мана је што даје добре резултате у случају постојања само једног објекта, а немоћан је код њиховог међусобног преклапања.



Слика 22: Оријентација троуглова и њихове нормале

У основи *back-front* алгоритма је идеја да све троуглове неког објекта треба раздвојити на две групе: на оне који су окренути ка посматрачу, и на оне који су окренути од посматрача. При моделовању тела потребно је водити рачуна да сви троуглови једног тела буду једнако оријентисани, својим нормалама ка спољашности тела. Уколико су ти услови испуњени, у *back-front* алгоритму се најпре нацртају троуглови чије нормале су усмерени на исту страну на коју посматрач гледа (јер се они налазе на "задњој" страни објекта) а након тога троуглови чије нормале су усмерене ка посматрачу (то су троуглови који се налазе на "предњој" страни објекта).

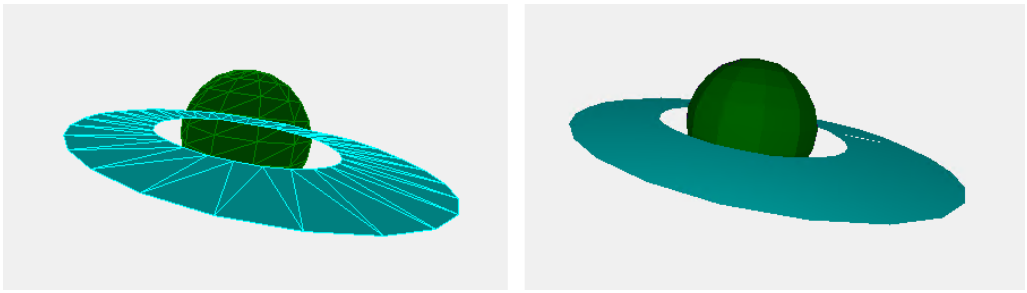
Којом страном је троугао окренут ка посматрачу (предњом или задњом) математички се одређује рачунањем скаларног производа вектора нормале тог троугла и вектора чији врх се налази у центру троугла а почетак у оку посматрача. Уколико је скаларни производ позитиван, поглед посматрача је усмерен на исту страну на коју и нормала троугла, односно ради се о троуглу који се налази са задње стране тела. Ако је скаларни производ негативан, ради се о троуглу који је са предње стране тела.



Слика 23: Исртавање троуглова коцке по редоследу дефинисања у моделу (лево) и по редоследу *back-front* алгоритма (десно). На обе коцке се могу уочити нормале троуглова, као и да недостаје најближе теме (услед *clipping*-а)

4.3. *Z-buffer* алгоритам

У случајевима када на сцени постоји више објеката који заклањају поглед посматрача на друге објекте, неоходно је установити редослед исцртавања сваког појединачног троугла од којих су сачињени.

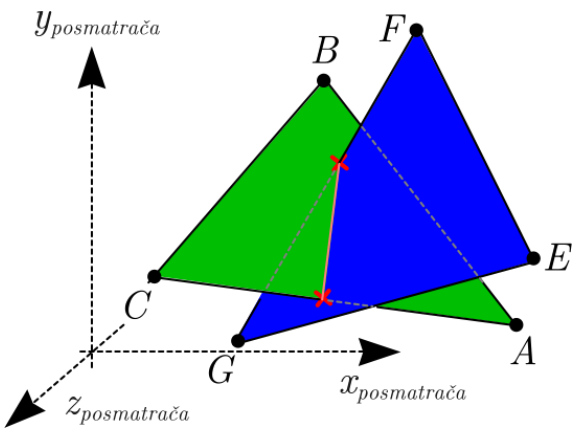


Слика 24: *Z-buffer* алгоритам омогућава правилно преклапање тела (десно)

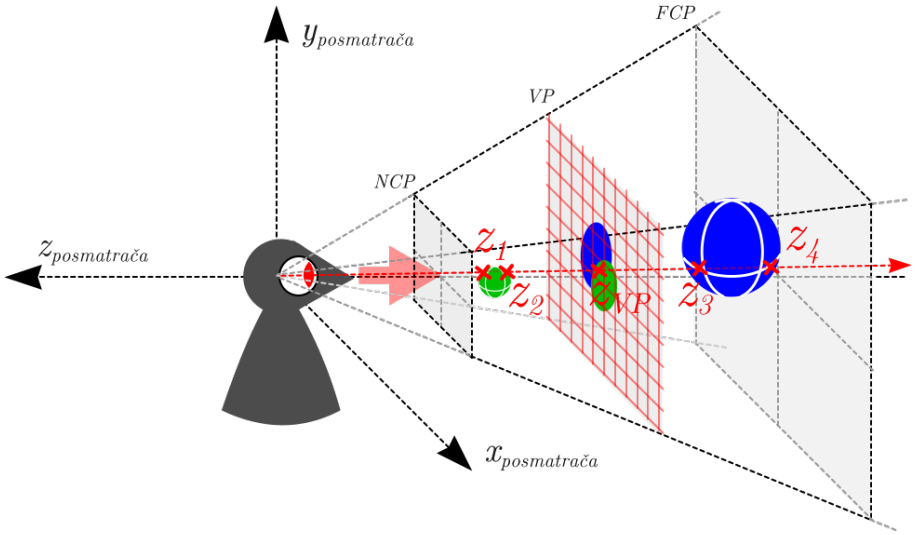
Редослед исцртавања троуглова зависи од удаљености тих троуглова од посматрача – троуглове који су ближе посматрачу треба исртавати након оних који су даље од посматрача. На тај начин ближи троуглови ће на слици преклопити даље троуглове – управо онако како посматрач и треба да их види. У овом раду нисам се бавио овим алгоритимима, већ сам тежиште рада ставио на значајно моћнији ***Z-buffer*** алгоритам.

Највећа предност *Z-buffer* алгоритма је што омогућава правилно приказивање чак и тела која су у колизији, односно која заузимају заједничку запремину. То значи да неки од троуглова тих тела могу да се секу, а алгоритам ће их правилно приказати. Мана овог алгоритма је што је спорији од других када се реализује софтверски, будући да се цртање обавља *пиксел по пиксел*.

У основи *Z-buffer* алгоритма је идеја да се за сваки пиксел екрана сачува додатни податак – колико далеко од посматрача се налази тачка коју посматрач види у том правцу, односно која је њена z-координата. Ти подаци за цео екран чувају се у одвојеној матрици у меморији која се зато зове *Z-buffer*.



Слика 25: За неке троуглове не може се одредити који је ближи посматрачу

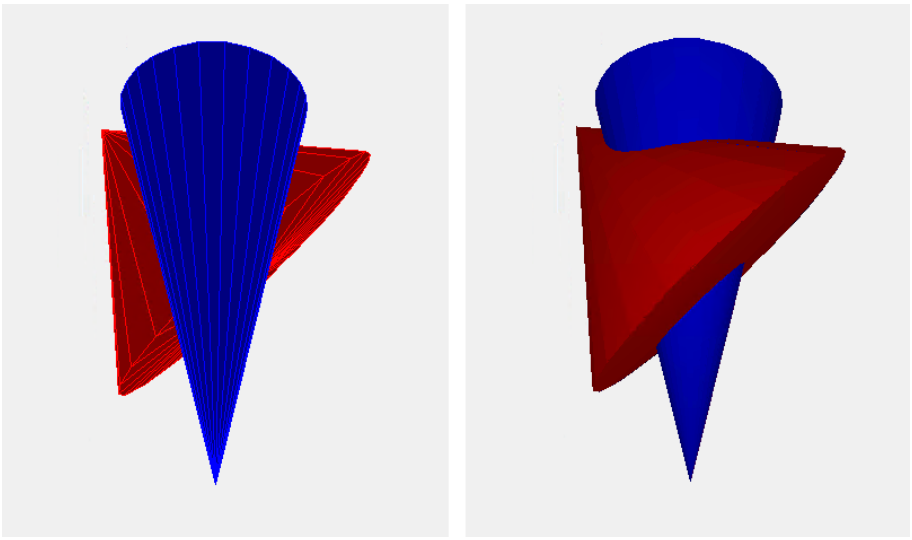


Слика 26: Прозор је подељен на сегменте који одговарају пикселима екрана – у сваком је потребно одредити који троугао је најближи посматрачу по z-оси, и потом обојити пиксел бојом тог троугла.

У *Z-buffer* алгоритму је пре исцртавања сваког појединачног пиксела неког троугла најпре потребно установити да ли је z удаљеност те тачке троугла мања од оне која је у том пикселу претходно нацртана. Уколико јесте, то значи да је нови троугао у тој тачки ближи посматрачу, па се тада пиксел попуњава бојом тог троугла а у *Z-buffer* се уписује његова удаљеност. У супротном, исцртавање текућег пиксела се прескаче.

4.4. Сенчење

Сенчење објекта је поступак којим се код посматрача ствара реалнија слика објеката у које гледа, јер се води рачуна о осветљености троуглова у зависности од положаја и врсте извора светла. У овом раду сам се бавио тачкастим изворима беле светлости као и дифузном изворoм светлости из свих праваца. При томе сам занемарио могућност заклањања извора светла од стране других објеката — нисам се бавио сенкама. Такође, сматрао сам да је површина свих

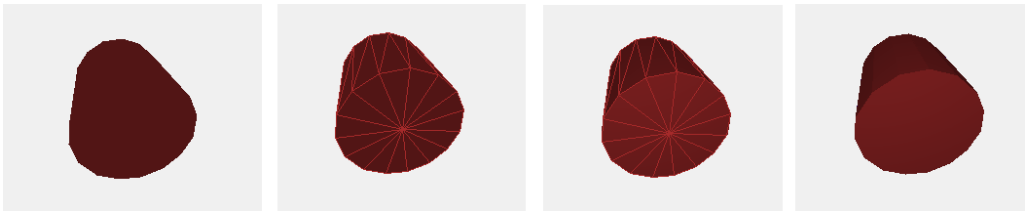


Слика 27: Колизија две купе – *Z-buffer* алгоритам омогућава правилно приказивање чак и тела која су у колизији (десно)

тела једнако глатка, као и да нема рефлексија (односно одраза слике једног објекта на површини другог). На крају, сматрао сам да су сви објекти једнобојни, односно нисам се бавио текстурама.

4.4.1. *Fixed-colour shading* алгоритам

Најједноставније могуће сенчење се постиже ***fixed-colour shading*** алгоритмом. Он се у суштини своди само на одређивање боје која је јединствена за цео троугао, и њено приказивање. Предност овог алгоритма је брзина, а мана што не даје верне слике тела. У случају да су сви троуглови неког тела једнообразно обојени а да се притом користи само дифузна светлост из свих праваца, овако осенчено тело ће изгледати потпуно "пљоснато", и може бити тешко закључити о ком телу се ради. Ситуацију донекле може да поправи истовремено приказивање и жичаног модела тела, али је то и даље далеко од реалне слике.



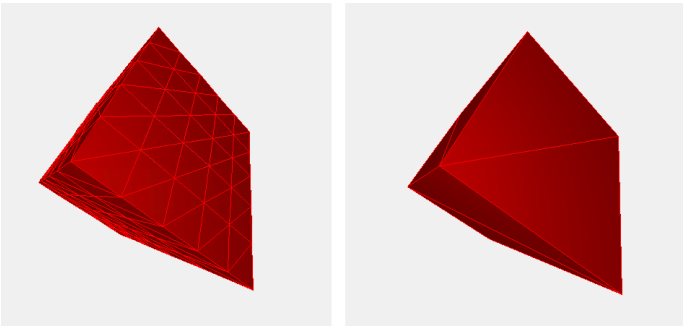
Слика 28: *Fixed-colour shading* алгоритам даје лоше резултате у случају дифузног светла из свих праваца (најлевља слика), што се донекле поправља додавањем жичаног модела (слика у средини лево), док у случају постојања тачкастог осветљења (две слике десно) ваљак се лакше препознаје

Бољи резултати се постижу уколико се на сцену уведе тачкасто осветљење. У том случају у *fixed-colour shading* алгоритму се у централну тачку троугла постави нормала, и израчуна упадни угао светлосног зрака из тачкастог осветљења. Што је тај угао ближи углу под којим у исту тачку упада и зрак из

ока посматрача, то боју троугла треба више посветлити. У обзир треба узети и храпавост површине троугла – што је храпавост већа то боју троугла треба још више посветлити. У циљу једноставности, у овом раду сам сматрао да се тачкасто осветљење налази у оку посматрача. На овај начин, чак и ако су сви троуглови тела једнообразно обојени у моделу, приликом приказивања на екрану ће неки од њих (у које посматрач директно гледа) бити светлији од других (који је налазе постранце у односу на посматрача), па ће слика изгледати реалније. Другим речима, троуглови ће бити приказани различитим нијансама исте боје, али ће сваки појединачни троугао у целини бити обојен на исти начин, па ће се (највише на местима додира тих троуглова) видети мане овог алгоритма.

4.4.2. *Flat shading* алгоритам

Напреднији сенчење се постиже ***flat shading*** алгоритмом. У његовој основи је идеја да се при исцртавању троугла на прозор примењује претходно описани алгоритам, али да се то не чини само за једну репрезентативну тачку троугла, већ за сваку појединачну тачку. На овај начин, иако је троугао једнообразно обојен, неке његове тачке ће бити светлије од других, што ће доприносити реалности слике, посебно у случајевима већих површина.



Слика 29: *Flat shading* алгоритам сваку тачку троугла коцке боји за нијансу другачијом бојом, што се најбоље уочава на троугловима великих површина (десно)

С обзиром на то да *flat shading* алгоритам троши додатно процесорско време за одређивање боје сваког појединачног пиксела, у циљу оптимизације овај поступак се примењује тек након што се (нпр. *Z-buffer* алгоритмом) установи који троугао треба исцртати у неком пикселу слике. И поред тога, при софтверској реализацији *flat shading* алгоритма може се приметити смањење *framerate*-а у односу на случај кад се он не примењује.

4.5. Могућа проширења

Једна од могућих идеја за убрзање рада је да се свако тело, поред описивања скупом троуглова, опише и координатама центра и полупречником, односно произвољном тачком на њему. Те две тачке би се, као представници целог тела, трансформационим матрицама пресликале у координатни систем посматрача, и потом израчунало да ли се тако пресликана сфера уопште сече

са пирамидом посматрања. Уколико не, сви троуглови тог тела би се изузели из процеса, и тиме значајно убрзао рад.

Идеја за даље унапређење програма има много. Нека од могућих проширења су:

→ *texture-mapping* – лепљење текстура, односно битмапа на површину тела;

→ *bump-mapping* – стварање привида неравнина на површини тела у циљу смањења броја троуглова који су потребни за моделовање ситних неправилности на телу; овај алгоритам представља проширење *texture-mapping* алгоритма;

→ *LogZ-buffer* – варијанта *Z-buffer*-а код које се чува логаритам вредности удаљености, тиме скоро потпуно елиминишући графичке артефакте (неправилности код исцртавања сцене) који настаје код тела која су близу једна другом а веома далеко од посматрача (познат као *Z-fighting*);

→ *A-buffer* – оптимизована варијанта *Z-buffer*-а која је способна за исцртавање тела која могу бити и прозирна (решавајући недостатак *Z-buffer*-а);

→ *anti-aliasing* – поправљање *aliasing*-а, односно оштрих ивица тела које настају приликом растеризације тела на екран (због ограниченог броја пиксела екрана);

→ *culling* – спречавање даље анализе тела (или његових делова) која су сакривена од погледа посматрача (али оптимизовано за огроман број тела на сцени);

→ *tesselation* – теселација, односно аутоматско партиционисање тела која су дефинисана као *Bezier*-ове површи на троуглове, у циљу омогућавања варијабилног броја троуглова у зависности од удаљености посматрача;

→ *shadows* – постављање реалистичних сенки на објекте делимично или потпуно заклоњене од извора светла, али видљиве од стране посматрача.

Наравно, наведене ставке само приближно одређују неке од проблема са којима се дизајнер 3D графичких библиотека сусреће.

5. Коришћење библиотеке

Главни циљ овог рада је био да се направи функционална 3D графичка библиотека, ради демонстрирања основних концепата рада, и, да не заборавимо, мог усавршавања у области 3D графике, која ме већ одавно интересује. Зато ћу у наставку објаснити како се библиотека користи, и које су њене могућности.

5.1. Апликација за приказ сцене

Да би коришћење библиотеке *Graphics3D* било могуће, било је неопходно направити апликацију за приказ сцене. Њена главна улога је да прикаже задати 3D модел на екрану, и омогући кориснику да утиче на тај приказ.

У било ком тренутку корисник може извршити следеће утицаје на приказ и анимацију:

→ да изабере модел који ће бити приказан;

→ да промени сложеност модела (повећа или смањи број троуглова којим се објекти моделују);

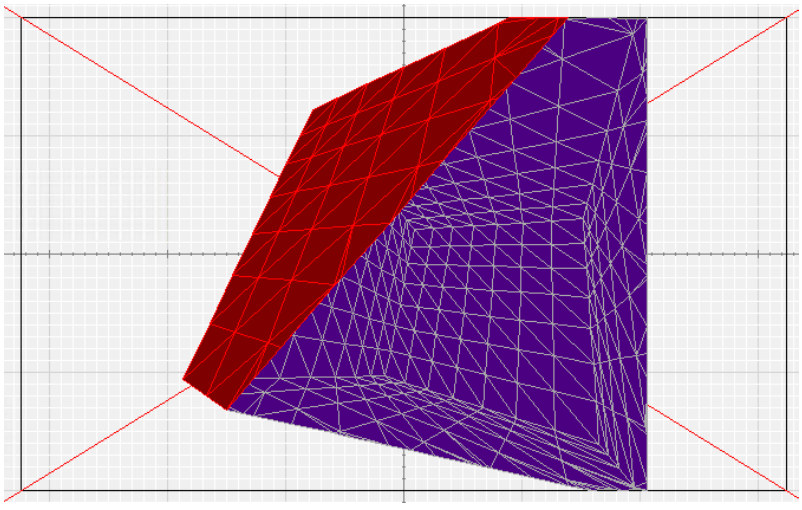
→ да утиче на брзину анимације, односно на проток времена симулације (аутоматски или ручно у унапред дефинисаним инкрементима);

→ да покреће посматрача транслаторно по свим осама и ротационо (у смислу усмеравања погледа лево-десно и горе-доле), и да га постави на неку од оса;

→ да бира које референтне ознаке унутар пирамиде посматрања ће бити приказане;

→ да одреди равни на којима ће се радити *clipping*;

→ да промени локацију NCP равни (приближи или је удаљи од посматрача, чиме је у могућности да "сецира" објекат);



Слика 30: Сецирање тела померањем NCP равни

→ да изабере начин исртавања модела (жичани или површински или оба истовремено);

→ да изабере алгоритам за одређивање дубине (*back-front* или *Z-buffer*);

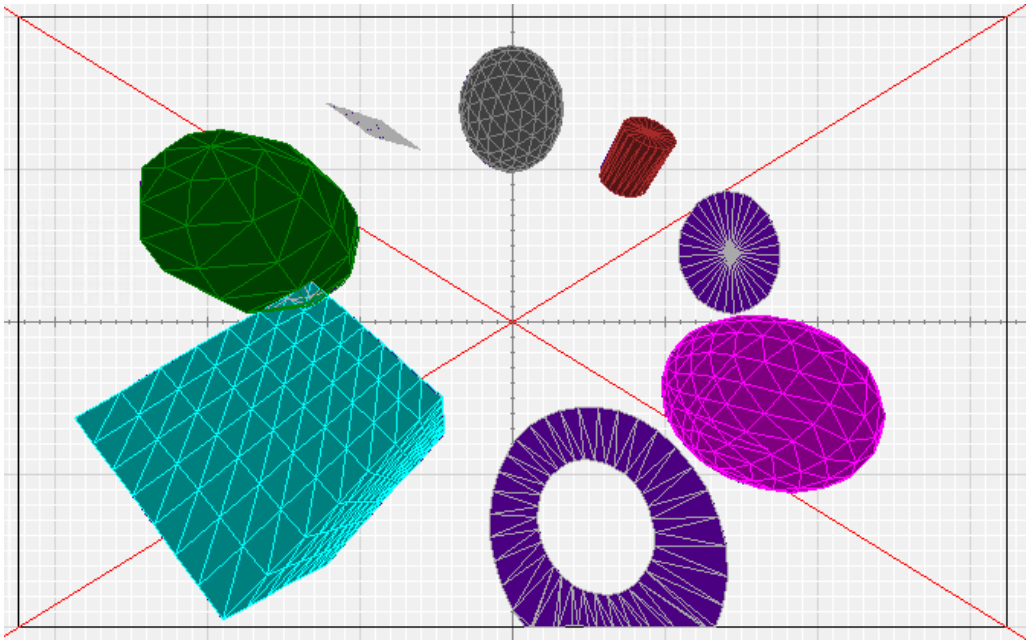
→ да изабере алгоритам за сенчење;

→ да укључи приказ нормала на троуглове.

5.2. Примери

5.2.1. Пример 1 – предефинисани облици тела

У *Graphics3D* библиотеци је на располагању неколико основних облика: коцка, лопта, ваљак и томе слично. Њихов (не)комплетан приказ дат је на наредној слици.



Слика 31: Основни геометријски облици на располагању у библиотеци *Graphics3D*

5.2.2. Пример 2 – тела која се секу

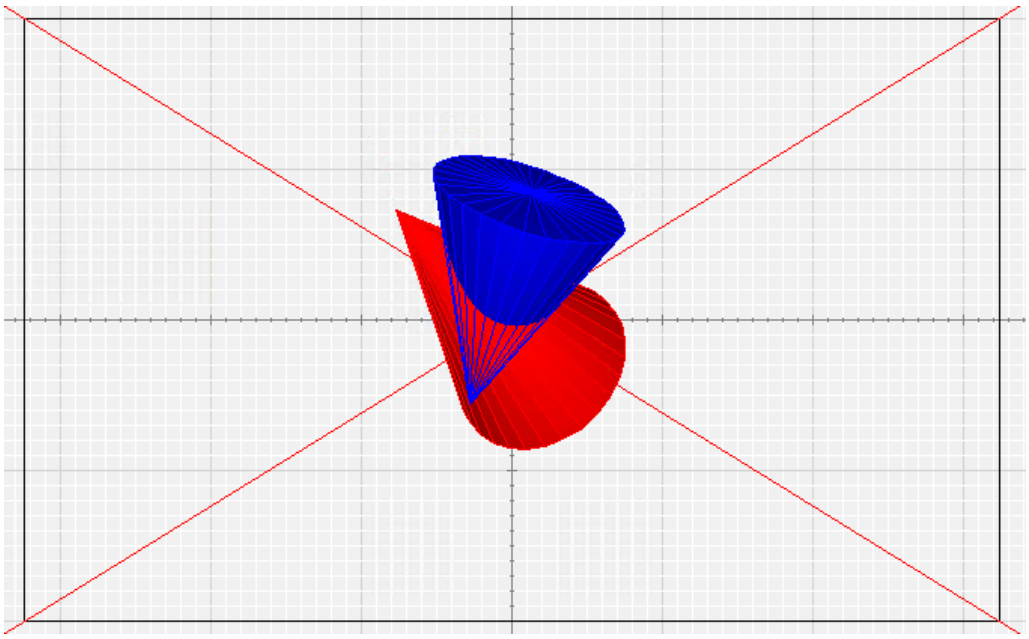
Да би се у апликацији за приказ сцене дефинисала сцена, потребно је у функцији `UpdateModels()` моделовати тела која ће се налазити на тој сцени.

```
scene.Add( Mesh.cone( 75, 150, Red,  3*detail, "kupa1" ) );
scene.Add( Mesh.cone( 50, 200, Blue, 3*detail, "kupa2" ) );
```

Поред тога, потребно је у функцији `UpdatePositions()` описати како се наведена тела крећу, односно описати како се рачунају трансформационе матрице тих тела. Наведене матрице могу да зависе било од времена симулације `Tsim`, било од угаоних параметара `radx`, `rady` и `radz` на које утиче корисник у циљу промене положаја покретних тела.

```
if     ( scene[i].name == "kupa1" )
   scene[i].Mtransf = Matrix4D.rotate( 0, 0, Math.PI/4 )
                    * Matrix4D.transl( 0, -150 * Math.Cos(radx), 0 );
else if( scene[i].name == "kupa2" )
   scene[i].Mtransf = Matrix4D.transl( 0,  100 * Math.Cos(radx), 0 )
                    * Matrix4D.rotate( Math.PI, 0, 0 );
```

У горе наведеном примеру, купа број 1 је најпре измештена по y-оси у косинусној зависности од -150 до $+150$ јединица након чега је закошена за $45°$, што значи да се креће горе-доле по y-оси, док је купа број 2 најпре закошена за $45°$ по x-оси, након чега је измештена по тој закошеној оси у косинусној зависности од -100 до $+100$ јединица, што значи да ове две купе у неком тренутку пролазе једна кроз другу.



Слика 32: Купа 1 и купа 2 које пролазе једна кроз другу током анимације

5.2.3. Пример 3 – Сунчев систем

Да бих илустровао могућности библиотеке *Graphics3D* у апликацији за приказ сцене направио сам модел Сунчевог система. Најпре сам направио веран модел, у смислу да су све димензије небеских тела, нагиби у односу на еклиптику, брзине ротације око своје осе, пречници орбита и брзине ротације око Сунца и других матичних планета одговарале стварности.

У функцији `UpdateModels()` моделовао сам следећа тела:

```
scene.Add( Mesh.uvsphere    (    109.125, Yellow,    10*detail, "Sunce" ) );
scene.Add( Mesh.uvsphere    (      0.382, DarkOrange, 3*detail, "Merkur") );
scene.Add( Mesh.uvsphere    (      0.949, Red,        5*detail, "Venera") );
scene.Add( Mesh.uvsphere    (      1.000, Blue,       5*detail, "Zemlja") );
scene.Add( Mesh.uvsphere    (      0.273, Gray,       2*detail, "Mesec" ) );
scene.Add( Mesh.uvsphere    (      0.530, Brown,      4*detail, "Mars"  ) );
scene.Add( Mesh.uvsphere    (     11.200, OrangeRed,  9*detail, "Jupiter") );
scene.Add( Mesh.uvsphere    (      9.410, Orange,     9*detail, "Saturn") );
```

```
scene.Add( Mesh.hollowcircle( 2.00*9.410,
                              1.35*9.410, Orange,     9*detail,
    "Saturnovi_prstenovi" ) );
scene.Add( Mesh.uvsphere    (      3.980, Cyan,       7*detail, "Uran"  ) );
scene.Add( Mesh.uvsphere    (      3.810, DarkCyan,   7*detail, "Neptun") );
```

Матрице трансформације које су потребне за позиционирање небеских тела и анимацију њиховог кретања израчунао сам у функцији `UpdatePositions()` на следећи начин:

```
if     ( scene[i].name == "Sunce"               )
   scene[i].Mtransf = rotateY(      0       ) * transl( 0, 0,  0.00 )
                    * rotateX(  7.25/180*PI ) * rotateY( 365.5*Tsim/2/PI/  25.38
    );
else if( scene[i].name == "Merkur"              )
   scene[i].Mtransf = rotateY( Tsim / 0.241 ) * transl( 0, 0,  0.38 )
                    * rotateX(  0.00/180*PI ) * rotateY( 365.5*Tsim/2/PI/  58.64
    );
else if( scene[i].name == "Venera"              )
   scene[i].Mtransf = rotateY( Tsim / 0.615 ) * transl( 0, 0,  0.72 )
                    * rotateX(177.30/180*PI ) * rotateY( 365.5*Tsim/2/PI/-243.02
    );
else if( scene[i].name == "Zemlja"              )
   scene[i].Mtransf = rotateY( Tsim / 1.000 ) * transl( 0, 0,  1.00 )
                    * rotateX( 23.44/180*PI ) * rotateY( 365.5*Tsim/2/PI/   1.00
    );
else if( scene[i].name == "Mesec"               )
   scene[i].Mtransf = rotateY( Tsim / 1.000 ) * transl( 0, 0,  1.00 )
                    * rotateX( 23.44/180*PI ) * rotateY( 365.5*Tsim/2/PI/   1.00
    )
                    * rotateY( Tsim /13.360 ) * transl(       60.00 )
                    * rotateX(  0.00/180*PI ) * rotateY( 365.5*Tsim/2/PI/  27.32
    );
else if( scene[i].name == "Mars"                )
   scene[i].Mtransf = rotateY( Tsim / 1.881 ) * transl( 0, 0,  1.52 )
                    * rotateX( 25.19/180*PI ) * rotateY( 365.5*Tsim/2/PI/   1.03
    );
else if( scene[i].name == "Jupiter"             )
   scene[i].Mtransf = rotateY( Tsim /11.863 ) * transl( 0, 0,  5.20 )
                    * rotateX(  3.12/180*PI ) * rotateY( 365.5*Tsim/2/PI/   0.41
    );
else if( scene[i].name == "Saturn"              )
   scene[i].Mtransf = rotateY( Tsim /29.447 ) * transl( 0, 0,  9.54 )
                    * rotateX( 26.73/180*PI ) * rotateY( 365.5*Tsim/2/PI/   0.44
    );
else if( scene[i].name == "Saturnovi_prstenovi" )
   scene[i].Mtransf = rotateY( Tsim /29.447 ) * transl( 0, 0,  9.54 )
                    * rotateX( 26.73/180*PI ) * rotateY( 365.5*Tsim/2/PI/   0.44
    );
else if( scene[i].name == "Uran"                )
   scene[i].Mtransf = rotateY( Tsim /84.017 ) * transl( 0, 0, 19.22 )
                    * rotateX( 97.86/180*PI ) * rotateY( 365.5*Tsim/2/PI/  -0.72
    );
else if( scene[i].name == "Neptun"              )
   scene[i].Mtransf = rotateY( Tsim /164.791) * transl( 0, 0, 30.06 )
                    * rotateX( 29.58/180*PI ) * rotateY( 365.5*Tsim/2/PI/   0.67
    );
```

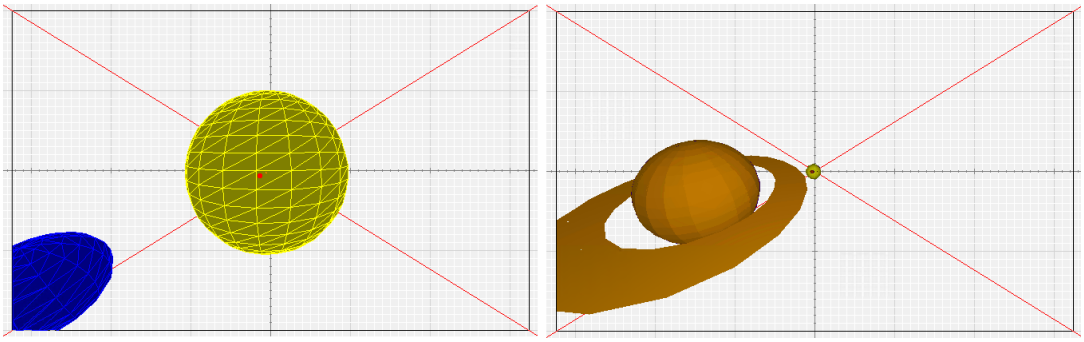
Трансформационе матрице за сва небеска тела рачунају се на исти начин. Ако посматрамо горе наведене инструкције са десна на лево, уочавамо да се:

1. најпре изврши ротација тела око своје осе (која је тада паралелна са y-осом света) у складу са протоком времена симулације `Tsim`,
2. затим се ротацијом око осе x нагне оса тела у односу на еклиптику,
3. потом се тело транслаторно измести по z-оси у складу са полупречником његове ротације око Сунца,
4. на крају се тело заротира око Сунца за адекватан угао у складу са протоком времена симулације `Tsim`.

Интересантно је уочити да се за Месец, Земљин сателит, трансформационе матрице рачунају у две фазе. Најпре се прве четири трансформационе матрице израчунају у смислу кретања Месеца у односу на Земљу – којим се опише ротација Месеца око своје осе, нагиб осе Месеца у односу на раван ротације око Земље, удаљеност Месеца од центра Земље и на крају ротација Месеца око Земље. Да би се Месец кретао заједно са Земљом, потом се примене исте четири трансформационе матрице које се користе за Земљу.

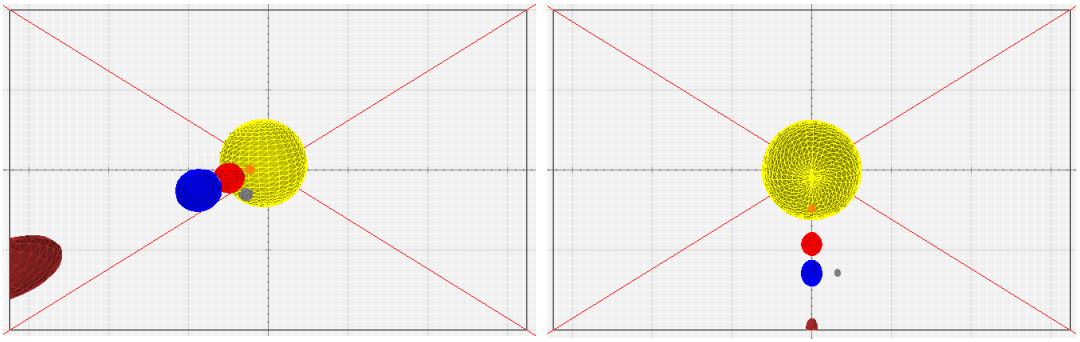
Треба такође уочити да су у почетном положају све планете нанизане дуж z-осе.

Испоставило се да је овако припремљен модел јако "празан", односно да у већини простора нема тела - јер су планете значајно мање од Сунца а притом јако удаљене од њега (као и међусобно). Сваки покушај позиционирања посматрача да се на екрану виде бар две планете истовремено, резултуовао је неуспехом - даља од те две планете била је приказана само као тачка. Чак и када је пречник орбита смањен 100 пута, сцена је била и даље врло "празна".



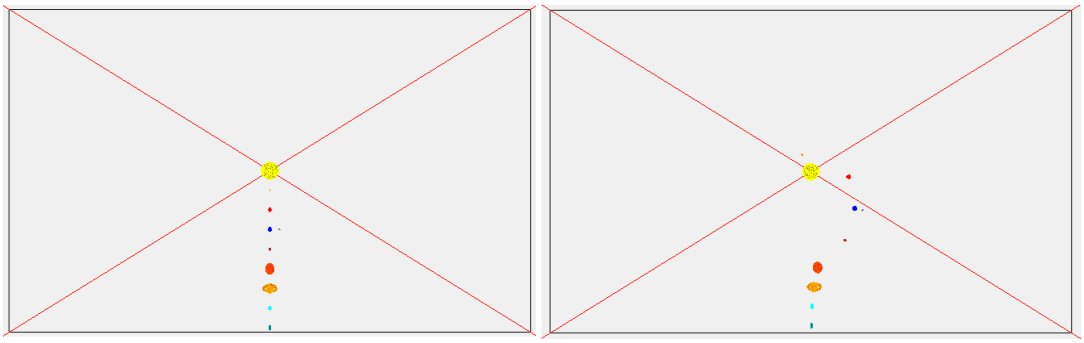
Слика 33: 100 пута "згуснут" Сунчев систем је и даље врло "празан" – углавном се види Сунце, и највише пар планета испред њега (лево – плава Земља, црвена Венера, десно – браон Сатурн, црвени Марс – тачка испред Сунца)

Зато сам величину малих планета и Месеца повећао 25 пута а великих планета 5 пута, док сам величину Сунца оставио незимењеном.

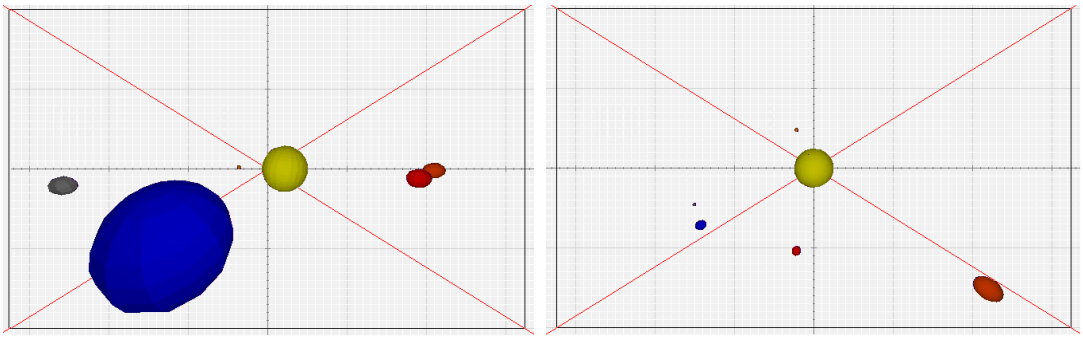


Слика 34: Повећање величине планета у односу на Сунце је приближније лаичком поимању изгледа Сунчевог система – наранџасти Меркур, црвена Венера, плава Земља, сиви Месец, браон Марс (лево – поглед из правца близу z-осе, десно – поглед из правца y-осе)

С' обзиром на то да су планете у близини Сунца близу једна другој, да бих још побољшао приказ, планете сам поставио на еквидистантне орбите. Такође, брзину ротације планета око Сунца убрзао сам 24 пута у односу на брзину ротације око сопствене осе.



Слика 35: Сунчев систем са еквидистантним орбитама планета је погодан за анализу ротације планета око Сунца – поглед са јако велике удаљености из правца y-осе (лево – пре почетка симулације, десно – након протока 60 дана од почетка симулације)



Слика 36: Поглед на Сунчев систем из више углова (лево – поглед ка Сунцу из близине Земље, десно – распоред планета у истом тренутку посматран из правца y-осе)

Након увођења наведених промена у реални модел Сунчевог система, његов приказ је постао изузетно интересантан за посматрање. Уочавање ротације небеских тела око своје осе је било олакшано приказивањем жичаног модела преко површинског, након правилног избора начина теселације лопте – тако да распоред троуглова модела омогући лако уочавање полова небеских тела.

6. Закључак

3D графика је јако широка област рачунарства, у којој је за кратко време постигнут брз и значајан напредак. Оно што се пре коју деценију сматрало за визуелно ремек-дело је из данашње перспективе исувише једноставно и наивно. Тако је и са другим технологијама, једноставно зато што се наше знање непрекидно мења и продубљује. Имајући то у виду, можемо са сигурношћу рећи да ће се у блиској будућности ствари у 3D графици радити другачије – ако квантни рачунари заживе, можда ће се исцртавање вршити помоћу њих, с обзиром на њихове предвиђене карактеристике попут изражене паралелизације процеса. Гледано из будућности, можда бележење координата тачака као 4D вектора уопште нема смисла због другачијих (квантних?) рачунара који ће тада бити у употреби, па ће изгледати као да смо кренули странпутицом још на почетку реализације овог матурског рада.

Једно је сигурно: 3D графика је начинила пробој и дубоко утицала на савремени начин рада – ствари су коначно могле да се раде једноставније и визуелно далеко јасније. На пример, радник у фабрици је и пре него што обради материјал, могао из свих углова да сагледа 3D модел предмета који верно приказује како ће тај предмет изгледати кад буде завршен. Архитекте су сада коначно могле да виде дигитални 3D модел свог рада, било да је у питању кућа, вишеспратница или читав кварт, и да крећући се кроз њега одмах уоче и отклоне све недостатке. Пројектанти су могли да виде како ће њихова машина радити и пре него што је употребљен и један шраф за њено састављање. Астрофизичари су сада могли да виртуелизују галаксије и да тестирају своје претпоставке над 3D моделима универзума. Редитељи су добили алтернативу која је изгледала много природније и штедела аниматорима године мукотрпног рада приликом прављења сцене. Све ово се одвијало паралелно са развојем нових и јачих графичких картица.

Слика 37: Принцеза Нејтири, дигитални лик из филма Аватар (2009)

Данас су се појавиле многе технологије на бази софтверске 3D графике (*Google Glass*, *Microsoft Hololens*, *Occulus Rift*), које имају два различита приступа – да се корисник осећа као да се налази у виртуелној стварности тиме што види искључиво 3D графиком симулирани свет – виртуална реалност *VR*

(*Virtual Reality*); и да корисник и даље види стварни свет, али са софтверски придодатим елементима – појачана стварност *AR* (*Augmented Reality*). Оба ова приступа имају корена у потребама корисника, било да је у питању интензивнији доживљај у видео игрицама, што код корисника виртуелне реалности изазива невероватан осећај; или брза провера временске прогнозе, резултата дербија, цртања на 3D платну (односно по ваздуху просторије), коришћењем интуитивних покрета и фокусирањем видног поља на предмет посматрања. Заиста невероватно.

Током израде овог рада подсетио сам се и утврдио знања из области линеарне алгебре и донекле аналитичке геометрије, али сам увидео и њихове реалне примене. Захваљујем се мојим менторима Снежани Јелић и Андреју Ивашковићу на пруженој помоћи и усмеравању током израде овог матурског рада.

Главна мотивација овог рада је била демистификација основних принципа 3D графике. Иако сам више пута претходно рекао да је област 3D графике интересантна, мислим да та реч не објашњава добро еуфорију која наступи када део програмског кода који до тада није ништа радио почне да производи невероватне слике. Надам се да ће ова библиотека бити од користи свима онима који настављају своје обучавање или истраживање у овом правцу, а понајвише будућим матурантима које занима 3D графика и који би ову библиотеку хтели проширити новим садржајима.

7. Референце

[1] Хомогене координате:
https://en.wikipedia.org/wiki/Homogeneous_coordinates

[2] Ротациони координатни системи:
http://www.cabiatl.com/mricro/obsolete/graphics/3d.html

[3] Матрица ортографске и перспективне пројекције:
https://www.scratchapixel.com/lessons/3d-basic-rendering/
perspective-and-orthographic-projection-matrix/
opengl-perspective-projection-matrix

[4] 4D Матрице:
http://www.opengl-tutorial.org/beginners-tutorials/
tutorial-3-matrices/

[5] *Clipping*:
https://www.tutorialspoint.com/computer_graphics/
viewing_and_clipping.htm

[6] Одређивање видљиве површине:
https://www.tutorialspoint.com/computer_graphics/
visible_surface_detection.htm

[7] *Z-buffer*:
https://en.wikipedia.org/wiki/Z-buffering

[8] Avatar:
https://www.i-fink.com/real-life-avatar/

[9] Ascii Art:
https://www.openindiana.org/forth_menu_extras/

[10] Ascii Doom for Vax Vms:
http://stvax.chat.ru/doom1.htm

[11] Car Wireframe:
http://www.broughtonhotels.com/blog/wp-content/uploads
/2015/03/105944443.jpg

[12] Elite Dangerous:
http://5.9.105.77/games/action/elite-dangerous-
horizons-best-games-sci-fi-space-open-world-8433.html

[13] Windows 3.11:
http://geek.corkymcgraw.com/wp-content/uploads/2015/09/
windows311.jpg