

|  |  |
| --- | --- |
|  | UNIVERZITET U NIŠU ELEKTRONSKI FAKULTET |

**Implementacija grafa trodimenzionalne scene kao dela alata za razvoj video-igara**

Diplomski rad  
 Studijski program: Elektrotehnika i računarstvo

Modul: Računarstvo i informatika

|  |  |
| --- | --- |
| Student: | Mentor: |
| Nikola Zlatkov, br. ind. 16593 | Aleksandar Dimitrijević |

Niš, <mesec> 20xx. godina

Univerzitet u Nišu Elektronski Fakultet

**Implementacija grafa trodimenzionalne scene kao dela alata za razvoj video-igara**

**3D Scene Graph implementation as a tool for video game development**

Diplomski rad  
 Studijski program: Elektrotehnika i računarstvo

Modul: Računarstvo i informatika

Student: Nikola Zlakov br. ind. 16593

Mentor: Aleksandar Dimitrijević

Zadatak: *Upoznati se sa arhitekturom alata i okruženja za razvoj trodimenzionalnih video-igara, proučiti postojeća rešenja, projektovati i implementirati graf scene. Funkcionalnost grafa scene, kao osnovne komponente za rukovanje scenom i objektima u njoj, demonstrirati u zasebnoj aplikaciji, razvijenoj za tu potrebu.*

Datum prijave rada: xx.xx.xxxx

Datum predaje rada: xx.xx.xxxx

Datum odbrane rada: xx.xx.xxxx

|  |
| --- |
| Komisija za ocenu i odbranu: |
| 1. <Prof. / Doc dr Ime Prezime>, Predsednik Komisije |
| 2. <Prof. / Doc dr Ime Prezime>, Član |
| 3. <Prof. / Doc dr Ime Prezime>, Član |

Implementacija grafa trodimenzionalne scene kao dela alata za razvoj video-igara

#### SAŽETAK

*Napomena: Sažetak rada (na srpskom i engleskom) treba da ukratko prikaže master rad kandidata, pri čemu treba izbegavati opšte napomene i fraze. U sažetku treba* ***konkretno istaći šta je urađeno u radu, uz pomoć kojih metoda i šta su najvažniji doprinosi rada (praktični, teorijski).*** *Sažetak rada je približno do oko 500 reči, odnosno do maksimalno 1 stranice A4 formata. Na kraju se daje 4-6 ključnih reči ili izraza u vezi predmeta rada.*

Moderni tehnološki procesi zahtevaju složene načine pokretanja i upravljanja višemotornim pogonima. Decentralizovano upravljanje u ovim složenim sistemima ima veoma važnu ulogu zbog potrebe povećanja produktivnosti, kvaliteta i brzine proizvodnje, ekonomičnosti, kao i uštede električne energije. U radu se prvo izlažu načini funkcionalne i mehaničke povezanosti višemotornih pogona u cilju sistematične analize i određivanja optimalne upravljačke konfiguracije. Zatim je definisan pojam decentralizovanog upravljanja i ukazano je na prednosti korišćenja ovog načina upravljanja u odnosu na centralizovano upravljanje. Kao primer jednog višemotornog pogona sa elastičnom mehaničkom vezom pogonskih vratila, koji se najčešće sreću u industriji, opisan je višemotorni pogon sistema za premotavanje. Data je principijelna šema, opisan je način funkcionisanja, a zatim je izvršeno njegovo detaljno matematičko modelovanje. Sistemi za premotavanje imaju veoma značajnu primenu u industriji za proizvodnju i obradu papira, gume, metala, plastičnih folija, tekstila i drugih tankih materijala sa elastičnim svojstvima. Sistemi za premotavanje se generalno sastoje iz istih elemenata, bez obzira na to o kakvim se proizvodima radi. Nezavisno od faze tehnološkog procesa, potrebna je visoka tačnost regulisanja brzine, a u toku proizvodnje potrebna je i visoka tačnost kod regulacije sile zatezanja trake.

Glavni cilj ovog rada je da se projektuje uspešan upravljački sistem višemotornog pogona sistema za premotavanje. Za ostvarivanje ovog zahteva korišćen je upravljački sistem koji je zasnovan na regulacionim petljama sa PI regulatorima. Podešavanje PI regulatora je veoma intuitivno, jednostavno i prihvaćeno u mnogim aplikacijama. Sa PI regulatorima se može postići zadovoljavajući kompromis u performansama u pogledu brzine odziva sistema, robusnosti i stabilnosti. Na osnovu matematičkog modela višemotornog pogona sistema za premotavanje napravljen je simulacioni model u programskom paketu MATLAB/Simulink i izvršeno je snimanje važnih procesnih veličina pomoću kojih su utvrđena teoretska razmatranja. Sagledavanjem rezultata dobijenih računarskom simulacijom, može se zaključiti da je primenjeni postupak decentralizovanog upravljanja sa PI regulatorima linijskih brzina i sila zatezanja višemotornog pogona sistema za premotavanje veoma efikasan jer se dobijaju zadovoljavajuće performanse i stabilnost sistema, uprkos promeni dinamike tokom procesa premotavanja. Linijske brzine kretanja trake i sile zatezanja trake se regulišu na adekvatan način u unapred dozvoljenom opsegu za vreme rada sistema za premotavanje, što se zahteva tehnološkim procesom.

**Ključne reči**: višemotorni električni pogoni, decentralizovano upravljanje, sistem za premotavanje, PI regulator.

3D Scene Graph implementation as a tool for video game development

#### ABSTRACT

*Napomena: Za ovaj deo rada korisno je uključiti opciju provere pisanja za UK English (Spelling and Grammar) iz Tools menija!*

Modern technology processes require complex algorithms for speed and torque control of multi-motor drives. Decentralized control of these complex systems has a very important role because of the need to increase productivity, quality and speed production, cost- effectiveness, and energy savings. In this work are first presented types of functional and mechanical connections between drives within the multi-motor drives in order to systematic analysis and determine the optimal control scheme. Then it is defined the concept of decentralized control and also are pointed out the advantages of using this method of control in relation to centralized control. As an example of multi-motor drives with elastic connection between drives, it is described the multi-motor drive of web winding system. It is given principle scheme, described way of functioning and then is done mathematical modeling of multi-motor drive of web winding system. Multi-motor drive of web winding systems have a very important role in the industry for manufacturing and processing of paper, metal, plastic film, textiles and other thin materials with elastic properties. Web winding systems are generally consisted of the same machine elements in spite of the diversity of the transported products. Regardless of the stage of the technology process, high accuracy of velocity and tension control for web is required.

The main aim of this work is to design a successful decentralized control system for multi-motor drive of web winding system. For the achievement of this requirement, it was used control system which is based on a control loop with PI controllers. Tuning of PI controllers is very intuitive, simple and accepted in many applications. With PI controllers can be achieved a satisfactory compromise in performance in terms of speed of system response, robustness and stability. Simulation model of multi-motor drive of web winding system it was made in software MATLAB/Simulink based on the mathematical model, and then were done set of simulations. Reviewing the results, obtained by computer simulation, it can be concluded that the applied method of decentralized control with PI controllers of linear speed and tension for multi-motor web winding system is very efficient because the system is stabile, despite the fact of changing dynamics during the winding/unwinding process. Velocities and tensions of web are regulated on adequately way in allowable range during the winding/unwinding process, that is required by technology process.

**Keywords**: multi-motor electrical drives, decentralized control, web winding system, PI controller.

**SADRŽAJ**

Contents

[1 UVOD 7](#_Toc87976759)

[1.2 Računarska grafika 7](#_Toc87976760)

[1.3 Game engine 7](#_Toc87976761)

[1.4 Istorija engine-a 8](#_Toc87976762)

[1.5 Komercijalni engine-i 8](#_Toc87976763)

[1.5.1 Quake porodica 8](#_Toc87976764)

[1.5.2 Unreal porodica 9](#_Toc87976765)

[1.5.3 Unity porodica 9](#_Toc87976766)

[1.6 Slojevi engine-a 10](#_Toc87976767)

[1.6.1 Hardver 12](#_Toc87976768)

[1.6.2 Drajveri 12](#_Toc87976769)

[1.6.3 Operativni sistem 12](#_Toc87976770)

[1.6.4 SDK i Middleware 12](#_Toc87976771)

[1.6.5 Sloj platformske nezavisnosti 13](#_Toc87976772)

[1.6.6 Sistemi Jezgra Engine-a 13](#_Toc87976773)

[1.6.7 Menadžer resursa 14](#_Toc87976774)

[1.6.8 Engine za iscrtavanje (Rendering engine / Renderer) 14](#_Toc87976775)

[1.6.9 Alati za profilisanje i otklanjanje grešaka 17](#_Toc87976776)

[1.6.10 Kolizije i fizika 17](#_Toc87976777)

[1.6.11 Animacije 18](#_Toc87976778)

[1.6.12 Uređaji za ljudsku interakciju 18](#_Toc87976779)

[1.6.13 Audio sistem 19](#_Toc87976780)

[1.6.14 Mreže (Networking) 19](#_Toc87976781)

[1.6.15 Osnova za igru (Gamplay Foundations) 19](#_Toc87976782)

[1.6.16 Specifični podsistemi igara (Aplikativni sloj) 20](#_Toc87976783)

# 1 UVOD

## Računarska grafika

Računarska grafika je skup tehnika i sredstava čijom se primenom olakšava obrada grafičkih podataka. Bavi se generisanjem slika uz pomoć sprege računarskog hardvera i softvera. Grafika predstavlja najoptimalniji način za prenos informacija od računara ka čoveku.

Svakodnevno se susrećemo sa računarskom grafikom na našim pametnim uređajima, računarima, televiziji i drugim multimedijalnim medijumima. Omogućava nam da uživamo u multimedijalnom sadržaju kao što su slika i video, interakcija sa korisničkim aplikacijama koristeći grafički interfejs,  obrada slika i 3d modela, vizuelizacija podataka, igranje video igara, projektovanje i crtanje u profesionalnom okruženju. Ova oblast iako je dosta rasprostranjena i uživa široku primenu, dosta je zahtevna kada se radi o razvoju grafičkih rešenja i zahteva snažne grafičke procesore za njihovu obradu.

Preteča razvoja moderne računarske grafike je naučni napredak u oblasti elektrotehnike, elektronike i televizije koji se dogodio u prvoj polovini dvadesetog veka. Koristila se kao sredstvo za komunikaciju između mernog uređaja i čoveka. S obzirom da je čovek vizuelno biće ovo je bio najefikasniji način za deljenje informacija sa korisnikom. Kao i u svakoj oblasti vezanoj za računarstvo doprinos je dala primena grafike u vojne svrhe, naime razvoj radarskih tehnologija, napredna avijacija, i raketna nauka. Bilo je potrebno prikazati obimnu količinu informacija korisniku, što je dovelo do daljeg razvoja računarske grafike kao discipline.

Vremenom su se grafički uređaji komercijalizovali što je omogućilo veću pristupačnost, time i ubrzan napredak grafičkih tehnologija. Konkretno razvoj video igara imao je najveći uticaj na razvoj naprednih grafičkih tehnika i specijalizovanog hardvera i softvera za obradu grafičkih podataka. Dok je za prikaz unapred generisanog sadržaja kao što su slike i animacije bilo potrebno obraditi sirove podatke jednom i onda prikazivati gotov multimedijalni sadržaj, kod video igara potrebno je obraditi sirove podatke za prikaz više puta u sekundi u realnom vremenu, jer prikaz u svakom momentu zavisi od inputa korisnika i stanja u kome se nalazi video igra. Sa tadašnjim hardverom i softverom ljudi su morali da imaju obimno znanje u oblasti matematike i računarskih nauka, a pored toga su primorani da smišljaju kreativna rešenja koja bi istiskala najoptimalnije performanse iz dostupnog hardvera.

## Game engine

Razvojem video igara uočili smo da postoje komponente i sistemi koji su jako slični i koriste se u skoro svim video igrama slične prirode. Da bi se unapredio proces rada programera razvijaju se apstraktni i generalizovani alati, komponente i sistemi koji ubrzavaju i povećavaju kvalitet razvoja video igara. Takve komponente se razvijaju i usavršavaju godinama, time raste kvalitet proizvoda koji ih koriste i omogućava programerima i dizajnerima da se fokusiraju na samu igru, umesto da se bave specifičnostima same platforme.

Često je potrebno da među tim sistemima postoji hijerarhijska relacija. Tako da bi bilo praktično da su svi alati, sistemi i komponente objedinjeni u okviru istog razvojnog okruženja koje pokriva celokupni razvojni ciklus video igara. Teži se ka tome da takvo razvojno okruženje sadrži sve što je potrebno za razvoj video igara kako programeri ne bi imali potrebe da ga napuštaju i gube vremena na integraciju eksternih alata. Takvo razvojno okruženje nazivamo *game engine*.

*Game engine-i* imaju ogromnu ponudu gotovih rešenja za probleme koji se sreću u razvoju video igara. Njihova primena je u apstrahovanju kompleksnih funkcionalnosti koje zahtevaju znanje i iskustva u raznim naučnim disciplinama kao što su analitička geometrija, trigonometrija, linearna algebra, numerička matematika, dinamika, kinematika, dinamika fluida, arhitektura i organizacija računara, uključujući poznavanje discipline dizajniranja video igara.

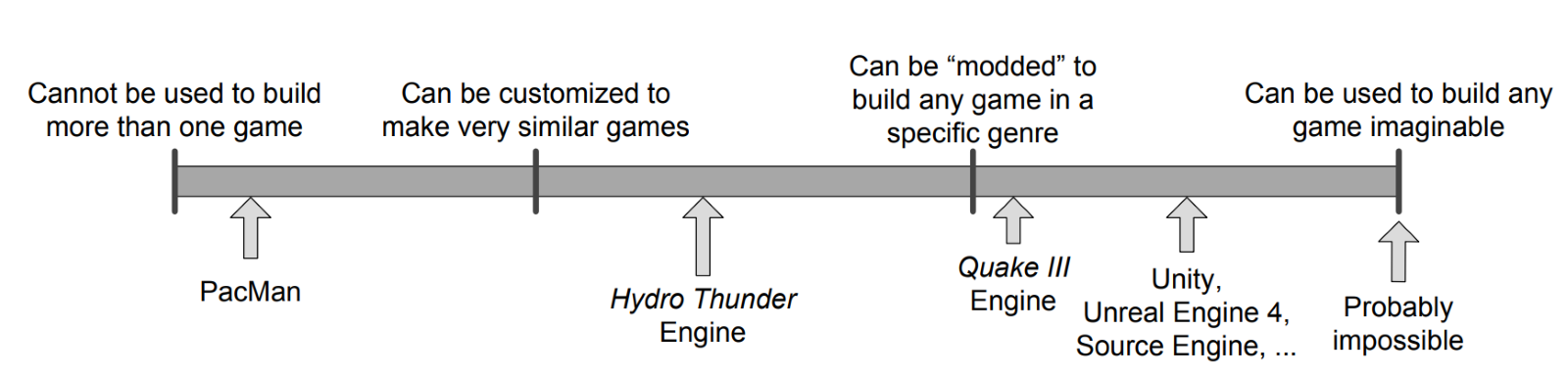
*Game engine-i* danas imaju za cilj rapidni razvoj video igara i namenjeni su za korisnike širokog spektra znanja i iskustva. Od početnika koji ne znaju puno o računarskoj grafici i žele da postepeno uče dok razvijaju proste video igre, do iskusnih programera kojima kojim nije dovoljno ono sto *engine* nudi, već modifikuju delove *engine-a* kako bi izvukli maksimalne performanse i omogućili prikaz fotorealistične grafike.

## Istorija engine-a

Termin *game engine* nastao je sredinom devedesetih godina kao posledica popularnosti igara iz perspektive prvog lica kao što je *Doom* koji je razvio *Id Software*. *Doom* je podelio komponente video igre na *core* komponente (jezgro)  (koga čini sistem za 3d grafički prikaz, sistem za detekciju kolizije, audio sistem, itd.) i aplikacione komponente (koje čine umetnički resursi, animacije, mape, pravila igre, zvučni efekti, muzika, itd.). Vrednost te podele se ogleda u tome što su proizvođači igara počeli da prodaju ta softverska rešenja, kako bi drugi proizvođači koristili gotovo jezgro igre (kasnije nazvan *game engine-om*) na kome razvijaju svoj aplikativni deo da bi dobili novi proizvod za mnogo manje uloženog novca i vremena.

Krajem 1990-ih, neke igre poput *Quake III Arena* i *Unreal* dizajnirane su sa ponovnom upotrebom jezgra i *modding-om*[[1]](#footnote-1) na umu. Napravljeni engine-i su veoma lako prilagodljivi za razne primene upotrebom jezika za skriptovanje kao što je Id-ev Quake C, a licenciranje engine-a počelo je da bude održiv sekundarni tok prihoda za programere koji su ih razvili. Proizvođači igara često licenciraju svoj engine klijentima koji prave video igre, a pored toga ga koriste koriste da prave svoje igre. Iako ova praksa i dalje uključuje značajna ulaganja u prilagođeni softver, takav biznis model je dosta ekonomičniji od razvoja engine-a igre od nule.

Presek između igre i engine-a je prilično malglovit i ne možemo sa sigurnošću tvrditi gde se nalazi. Međutim možemo lako dati zaključak da li igra može uz manje modifikacije postati engine na osnovu toga koliko je igra fleksibilna za ponovnu upotrebu na nekom drugom projektu. Ako je većina funkcionalnosti zakucana u kodu adaptacija rešenja za neku drugu igru postaje nemoguća, može se slobodno reći da se radi o igri koja nema odvojen engine.

**Slika 1** Spektar mogućnosti ponovne upotrebe engine-a

## Komercijalni engine-i

Devedesetih godina na tržištu se pojavljuju komercijalni game engine-i. Razlikuju se po stepenu specijalizacije i vrsti specijalizaciji. Engine namenjen za pucačine iz prvog lica dosta se razlikuje od engine-a za igre strategije. Međutim neka rešenja su generalizovana na višem nivou, tako da omogućavaju razvoj skoro svih žanrova igara, kao švajcarski nož game engine-a.

Zašto onda svi ne koriste generalizovana rešenja, ako su već toliko fleksibilna? Činjenica da se ovi engine-i snalaze u svakom polju, ne podrazumeva da se snalaze idealno. Naime uvek će korišćenje specijalizovanog engine-a proizvesti bolje rezultate u polju grafike, performansi i gameplay-a. Važi engleska izreka „Jack of all trades, master at none“ (poznavatelj svega, majstor ni iz čega).

### Quake porodica

Quake engine razvio je Id Software 1996. godine, za ravoj igre Quake. Prvenstveno je namenjen za iscrtavanje 3D grafike, učitavanje modela i tekstura, reagovanje na input sa tastature i miša i komunikaciju sa serverom preko mreže, preciznije namenjen za 3D pucačine iz prvog lica. Nakon izdavanja, engine se nadograđuje i koristi za razvoj igara Quake II i Quake III Arena, sa nazivom Quake II engine i id Tech 2, respektivno. Popularnost ove porodice engine-a ogleda se u ogromnoj količini izvedenih engine-a tokom njihovog životnog ciklusa. Trenutno je dostupan izvorni kod originalnog Quake i Quake II engine-a, engine je odlično struktuiran i može da služi kao osnova i dan danas.

U vremenu kada je on razvijen hardware je bio na dosta nižem nivou od danas, igre su pokretali procesori sa taktom od 50–70 MHz, dok je danas standard 3000–4000 MHz. Limitirani tehnologijom morali su da razviju revolucionarne tehnike za optimizaciju. Mape su se kreirale kao skup 2D primitivnih geometrijskih oblika, koji su se koristili kao četkica za iscrtavanje mape, koja se kasnije mapira u pravu 3D mapu koja se vidi na ekranu. Bilo je potrebno pripremiti senke unapred, naime senke su se „pekle“ direktno u teksture koje se iscrtavaju kao omotač modela. Koristi Z-buffer za dodatnu optimizaciju, odnosno za odbacivanje prekrivenih modela iz perspektive posmatrača kod iscrtavanja. Da bi se smanjio posao kod iscrtavanja, koristi se mehanizam za deobu mape na sekcije, tako da može da odbaci veliki deo mape koji nije trenutno vidljiv igraču. Bez ove optimizacije bilo bi potrebno izvršiti proveru da li se svaki model nalazi u vidokrugu igrača i da li je bilo koji od modela pokrio taj model. Za deobu mape na sekcije koristi se binarno particionisanje prostora (BSP), koje ubrzava pretragu traženog poligona. Struktura stabla omogućava odsecanje grana na kojoj se nalazi veliki broj primitiva koje nije potrebno razmatrati kod iscrtavanja. Obilaskom stabla generiše se polje potencijalno vidljivih poligona koji ulaze u obračun kod iscrtavanja. Problem ovog rešenja je što koristi veliku količinu memorije za obeležavanje poligona kao viljivim ili nevidljivim, zbog toga se koristi enkripcija dužine ponavljaja (RLE), jer se u kodiranju uzajamno ponavlja veliki broj istih znakova (1 ili 0).

### Unreal porodica

Epic Games 1998. godine izdaje pucačinu iz prvog lica pod nazivom „Unreal“. Od tada Unreal Engine postaje veliki konkurent Quake engine-u u žanru pucačina iz prvog lica. Pored toga UE se koristi za razvoj igara u raznim žanrovima, ali je pored toga video primenu i u filmskoj i televiziskoj industriji. Kroz životni vek prošao je kroz puno iteracija, i izdržao je test vremena tako što je uvek bio lider u svojoj industriji. Unreal engine je aktuelan i na dan pisanja rada i trenutna aktuelna verzija je UE 4, dok je UE 5 najavljen za 2022. godinu.

Prva generacija UE razvijena je od strane Tim Sweeney-a, osnivača Epic Games-a. UE se originalno oslanjao na softversko iscrtavanje grafike, odnosno kalkulacije vezane za iscrtavanje su se obavljale na procesoru. Vremenom omogućeno je korišćenje moći grafičkih uređaja namenjenih u te svrhe. Kao kontrast primitivnoj prvoj generaciji engine-a, novije iteracije omogućuju prikaz fotorealistične grafike kao što se vidi na slici 2.



**Slika 2** Poređenje grafike UE1 i UE5

### Unity porodica

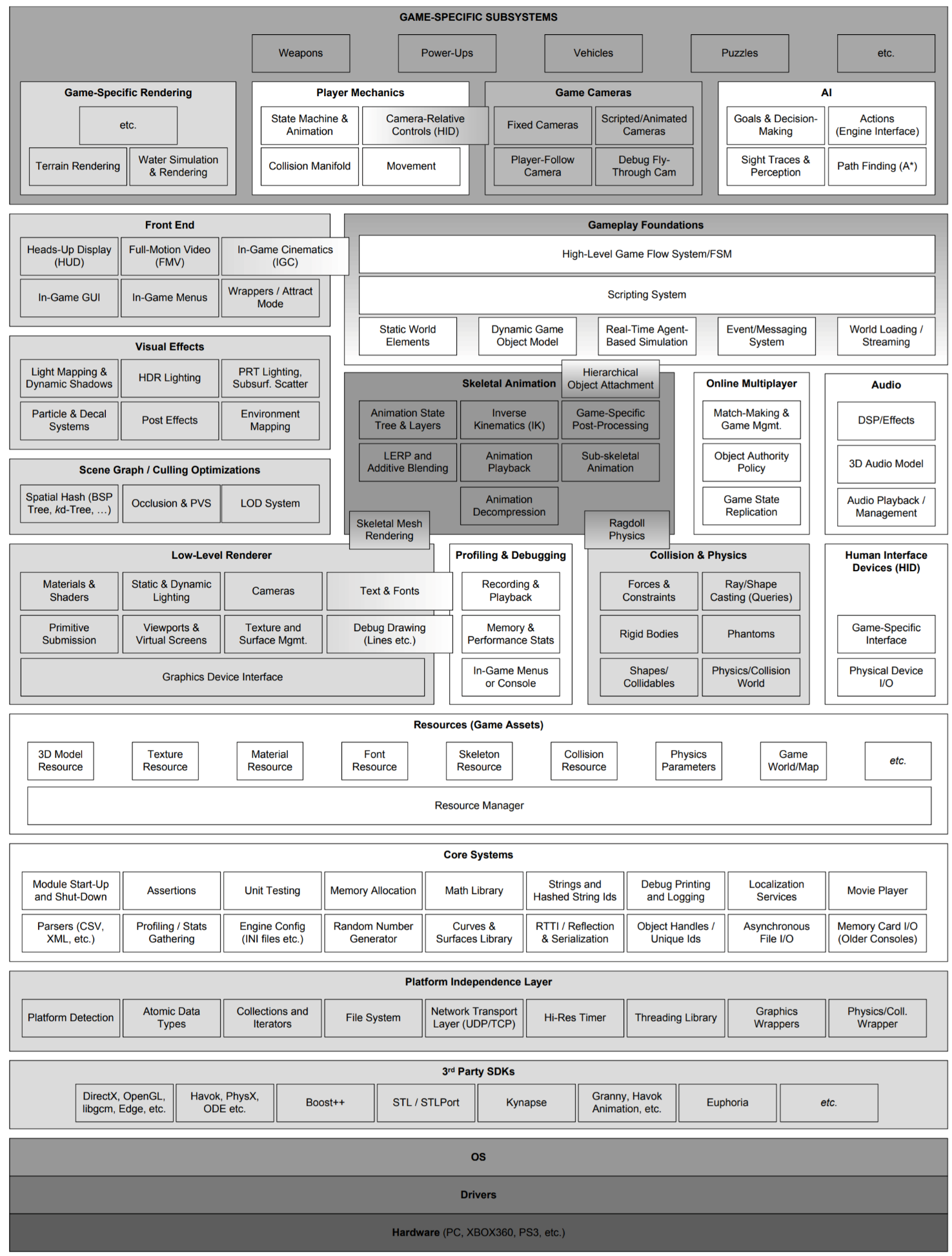
Unity je trenutno najpopularniji multiplatformski engine na tržištu. Za njegov uspeh zaslužan je nizak ulazni prag, linearna kriva učenja, mogućnost izdavanja igre na veliki broj platforma i naravno činjenica da se može koristiti besplatno za manje projekte. Količina podržanih platformi za koje može da se razvija igra je ogroman, obuhvata sve aktuelne konzole (sve playstation konzole nakon 3, sve Xbox konzole nakon 360, Nintendo Switch), desktop računare (Windows, Linux, MacOS), mobilne uređaje (Android, iOS) i web pretraživače (Chrome, Mozilla, Safari).

Cilj Unity-a je da olakša razvoj video igara i omogući početnicima da postepeno uče o programiranju video igara, što ga najčešće čini engine-om na kome ljudi uče da prave video igre. Tom cilju ide u korist grafički editor koji omogućava korisnicima lako ubacivanje i manipulacija objekata u sceni, a pritom sve što se ubaci u scenu iscrtava se u realnom vremenu. Unity nikad nije bio namenjen kao zamena za engine koji se koriste za pravljenje visokobudžetnih igrica, ali je zato odličan za igrice manjeg ili srednjeg obima.

## Slojevi engine-a

Dobar engine mora da ima modularnu strukturu kako bi njegovi moduli bili ponovo upotrebljivi i kako bi mogli da se razvijaju nezavisno jedan od drugog. Monolitne strukture su prihvatljivo rešenje za razvoj igara koje neće biti modifikovane u budućnosti, međutim to nije praksa zbog toga što su moderne igrice veoma kompleksne i razvoj rešenja od nule koje ne može biti ponovo upotrebljeno nema mnogo smisla.

Tipični moderan engine ima ogroman broj komponenti, koje su sastavljene od manjih srodnih komponenti. Komponente su podeljene u hijerarhijske slojeve, u kojima važi pravilo da viši slojevi zavise od nižih, nikako obrnuto. Ako niži nivo zavisi od višeg nastaje ciklična zavisnost koja povećava međusobnu povezanost i dovodi do softvera koji se teško testira, a lako nastaju greške.



**Slika 3** Arhitektura game engine-a

### Hardver

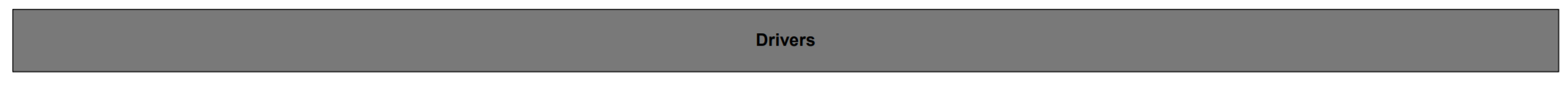
Hardverski sloj prikazan na slici x predstavlja sistem koji će pokretati igru na svojoj platformi. Neke od platformi su Microsoft Windows, Linux, Android, iOS, Playstation 5, Xbox Series X, Nintendo Switch itd.



**Slika 4** Hardverski sloj

### Drajveri

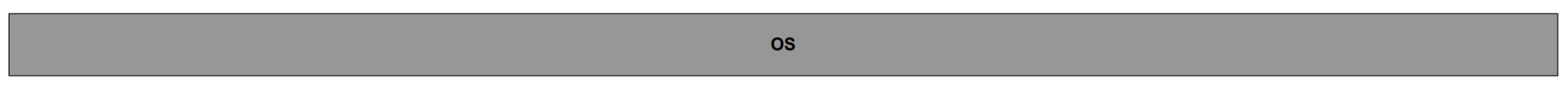
Drajveri su softver niskog nivoa koji obezbeđuje proizvođač operativnog sistema ili hardvera. Oni upravljaju hardverskim resursima i štite više slojeve engine-a od detalja upravljanja različitim hardverskim komponentama.



**Slika 5** Drajverski sloj

### Operativni sistem

Svaki kompleksniji uređaj ima operativni sistem koji upravlja hardverskim i softverskim resursima na višem nivou od drajvera. Zadužen je za dodelu hardverskih resursa (memorije i procesorskog vremena) procesima koji se izvršavaju na njemu, to jest omogućava konkurentno ili paralelno izvršenje procesa (naša igra je jedna od njih). To znači da je potreban kako bi na primer procesor na playstation konzoli pauzirao izvršenje igre da prikaže obaveštenje o pristigloj poruci. Naravno operativni sistem se koristi za mnogo više od toga, ali zadržaćemo se na tome.

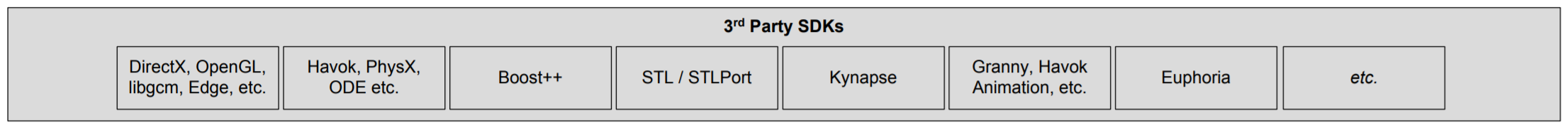


**Slika 6** Sloj operativnog sistema

### SDK i Middleware

Ovaj sloj nije obavezan u svakom engine-u, ali korišćenje gotovih biblioteka za delove engine-a ima mnogo prednosti. Produkcije koje razvijaju ove biblioteke najčešće su usko specijalizirane za to i rade na tome dugi niz godina. To znači da je taj softver prošao kroz sito i rešeto, i da će programerima ubrzati razvoj engine-a i garantovati stabilnije komponente.

Mana ovih rešenja je što su detalji implementacije najčešće skriveni, a korisnicima je dostupan samo interfejs kojim pristupaju funkcijama biblioteke. Takođe nije uvek jednostvno adaptiranje za potrebe svakog engine-a, što može da dovede do toga da je potrebno adaptirati podatke koje šaljemo toga pre poziva funkcije interfejsa, što dovodi do lošijih performansi i sklonosti pojavljivanja grešaka kod adaptacije funkcija.



**Slika 7** SDK i Middleware sloj

#### Algoritmi i strukture podataka

Video igre zahtevaju obradu velike količine podataka više puta u sekundi, što znači da je od suštinskog značaja da se ti obračuni izvršavaju na optimalan način. Zbog toga težimo da koristimo algoritme i strukture podataka koje su napravljene od strane stručnjaka, strogo testirane i bezbedne za upotrebu.

Međutim ta rešenja ne odgovaraju uvek za upotrebu u visokoperformantnom softveru kao što je game engine, pogotovo ako pravimo multiplatformsku igru, zbog toga što interno hardverske platforme rade različito na nižem nivou. Uglavnom je neophodno imati to na umu kada se radi za specifičnu platformu.

#### Grafika

Većina engine-a je izgrađena preko biblioteke grafičkog interfejsa. Oni služe za komunikaciju procesora sa grafičkim uređajem u sistemu. Zadužen je za prenos podataka i izvršenja grafičkih programa. Neke od poznatijih su:

* OpenGL (multiplatformski SDK za 3D grafiku)
* DirectX (konkurencija OpenGL-u koja je eksluziva na Windows-u)
* Vulkan (podržan je na puno platformi slično kao OpenGL, ali nudi funkcionalnosti na dosta nižem nivou kako bi se maksimalno optimizovao grafički engine)

#### Kolizije i fizika

Detekcija kolizija i dinamika krutih tela (poznatija kao fizika) ima razvijena gotova rešenja poput:

* Havok (industrijski standard)
* PhysX (popularan engine za fiziku razvijen od strane NVIDIA-e)
* Open Dynamics Engine (poznato open source rešenje)

#### Animacije

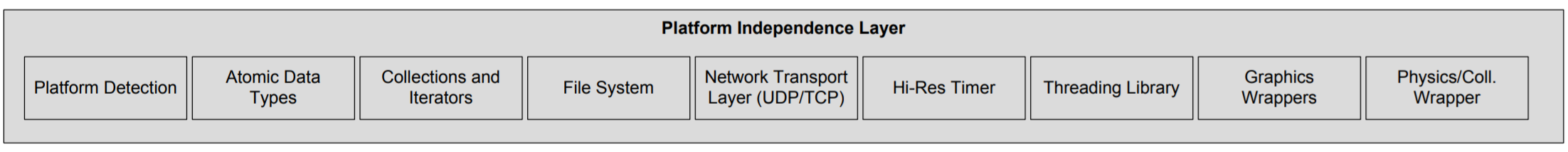
Postoji nekoliko odličnih gotovih rešenja za obradu animacija:

* Granny (veoma jednostavan alat za korišćenje, koji je zadužen za čitanje eksportovanih podataka modela i animacija, i kao snažan sistem za animacije)
* Havok Animation (u poslednje vreme teško je razlikovati šta je fizika, a šta animacija, tako da je Havok uzeo pod svoje da napravi alat koji je komplementaran sa Havok fizičkim alatom)

### Sloj platformske nezavisnosti

Većina engine-a su dovoljno fleksibilni da omoguće pokretanje na različitim hardverskim i softverskim platformama. Velike kompanije žele da pogode što veći deo tržišta svojim proizvodom, zbog toga su multiplatformski engine-i veoma privlačni u industriji video igara.

Ovaj sloj je zadužen da upakuje niže slojeve na takav način, da viši slojevi ne znaju koja je ciljana platforma za pokretanje igre. Programera koji razvija video igru ne treba da zanima način na koji se čitaju fajlovi, kako se osluškuje pomeraj miša i kako se hvataju sistemski događaji. Igra treba da funkcioniše konzistentno na svim platformama.

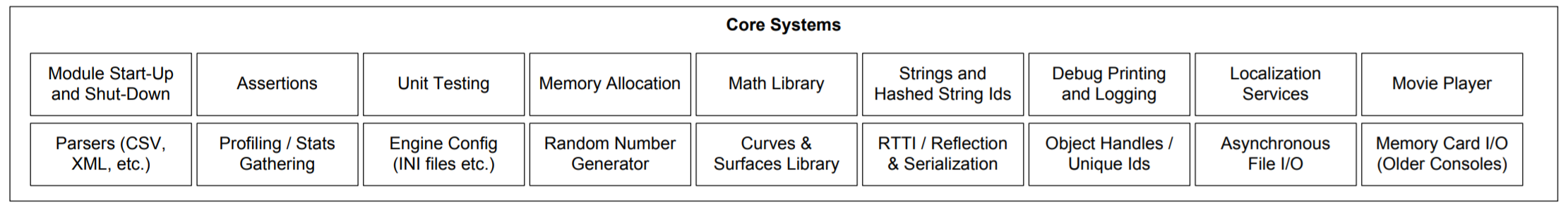


**Slika 8** Sloj platformske nezavisnosti

### Sistemi Jezgra Engine-a

Svakom engine-u je potreban skup praktičnih pomoćnih sistema koji će se stalno koristiti u višim slojevima game engine-a. Predstavljaju poslednji sloj apstrakcije, odnosno skrivanja implementacije od programera koji razvija video igre. Neki od delova ovog sistema su:

* Assertions (kod koji pomaže u razvoju kod provere nastanka logičkih grešaka; taj kod se izbacuje kada se produkciona verzija igre pošalje na tržište)
* Upravljanje memorijom (svaki engine ima svoj sistem za upravljanje memorijom, zato što ugrađena rešenja za alokaciju memorije i smeštanje podataka nisu optimalna za kreiranje video igara)
* Matematičke biblioteke (svaki engine mora da ima robustnu matematičku biblioteku koja nudi visoke performanse za ciljanu platformu)

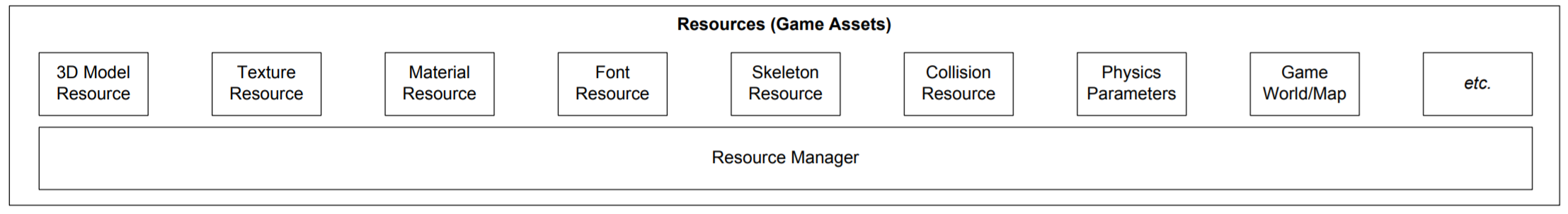


**Slika 9** Sloj jezgra engine-a

### Menadžer resursa

Svaki engine nudi svoje rešenje za upravljanje resursima. Video igre mogu da prelaze veličine iznad 100GB. Od ključnog je značaja optimalno upravljanje ogromnom količinom resursa.

Menadžer resursa je zadužen da čita teksture, šejdere, modele, skelete, animacije i skripte sa diska i smešta na radnu memoriju računara. Međutim ne može samo da učitava i gomila podatke, potrebno je da učita podatke na način na koji je optimalno njihovo čitanje i dealokacija. Potrebno je stalno izbacivanje resursa koji se više ne koriste i ubacivanje aktuelnih resursa na njihovom mestu.



**Slika 10** Sloj za upravljanje resursima

### Engine za iscrtavanje (Rendering engine / Renderer)

Definitivno najbitniji deo engine-a. Sve ovo do sada je bespotrebno ako na kraju ne iscrtamo nešto na ekranu. Pored toga što je najbitniji deo engine-a on je i najkompleksniji.

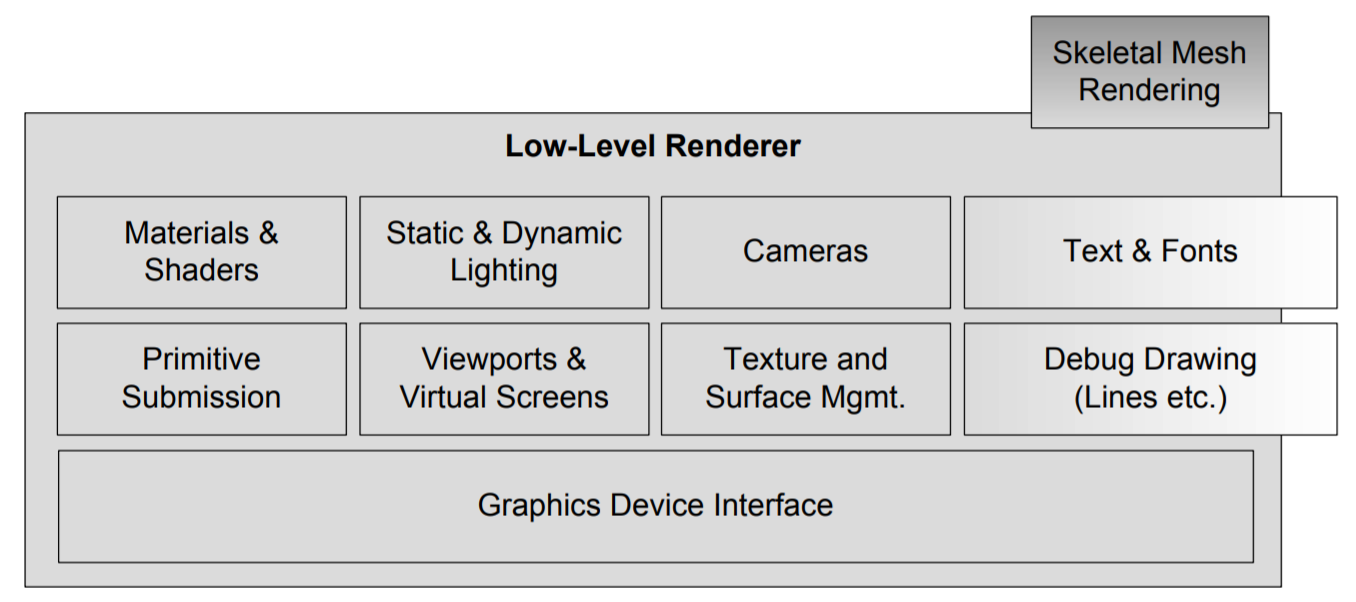
Postoje dve filozofije kod kreiranja ovog engine-a. Jedna je da postoji interfejs koji apstrahuje detalje implementacije i omogućava jednostavnije korišćenje preko interfejsa. Međutim postoje i engine-i za iscrtavanje koji su napravljeni na dosta nižem nivou i tako dozvoljavaju programeru veću kontrolu kod iscrtavanja sirovih podataka. Ovaj pristup ima bolje performanse, ali po ceni uvećanja posla i potrebnog znanja programera.

Zbog kompleksnosti engine-a za iscrtavanje, on se dalje deli u podslojeve.

#### Iscrtavanje na nižem nivou (Low-Level Renderer)

Low-level renderer ne brine o tome da li je potrebno iscrtati neki deo scene, već ima zadatak da što brže iscrta što veći broj geometrijskih primitiva.

Grafičke biblioteke zahtevaju velike količine koda za inicijalizaciju i pripreme pre iscrtavanja na ekranu. Zbog toga mora da postoji interfejs preko koga viši slojevi pozivaju funkcije low-level renderera.



**Slika 11** Low-Level Renderer

Ostale komponente u nižem nivou služe za enkapsulaciju podataka i obračunavanja koja su logički srodna. Ove komponente treba da komuniciraju međusobno, ali da ostanu slabo spregnute kako bi se smanjile međusobne zavisnosti koje izazivaju mnogo problema tokom razvoja. Podaci koje spremi svaka komponenta se objedinjuju i iscrtavaju na optimalan način.

Svaka komponenta učestvuje u iscrtavanju krajnje slike. Kamera definiše lokaciju posmatrača i pravca pogleda u sceni, dok materijal modela određuje njegovu osvetljenost. Oni moraju da kolaboriraju tako da se na ekranu model iscrta na određenom mestu u zavisnosti od toga gde se nalazi kamera i kako je okrenuta, dok senke i ambijentalno osvetljenje zavise od pozicije i pravca svetla, a odsjaj modela zavisi od materijala od koga je model, svetlosnih atributa i lokacije posmatrača.

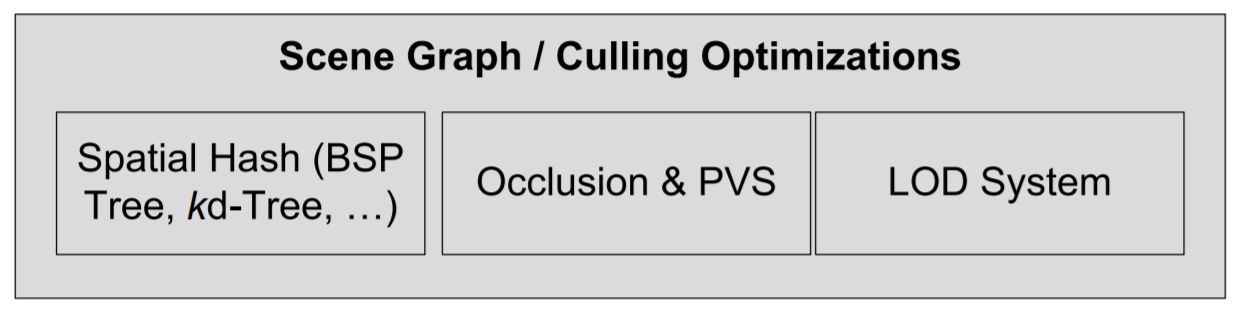
#### Graf scene / Odbacivanje primitiva koje nisu u vidokrugu

Pre iscrtavanja obavezno se odbacuju primitive koje neće biti vidljive na krajnjoj slici. Nije potrebno iscrtavanje primitiva koje su sakrivene iza drugih neprovidnih primitiva, lica modela koja nisu okrenuta ka nama, ili primitiva koje se nalaze van pogleda. Najjeftinija, a ujedno i najefikasnija optimizacija je odbacivanje ovakvih primitiva kako se ne bi ni razmatrale kod iscrtavanja.

Moguće je odraditi hijerarhijsku prostornu podelu scene. Ako se posmatrač nalazi u sobi nekog apartmana nije potrebno da se iscrta svaka soba, već samo ona u kojoj se nalazi ili deo susedne sobe ako je vidljiv kroz otvorena vrata. Hijerarhija bi pratila sledeću strukturu:

Apartman -> Soba -> Kompozitni nameštaj -> Posebni delovi nameštaja -> Primitive

Iscrtavanje se vrši odozgo nadole i vrši se odsecanje u zavisnosti od logike iscrtavanja. Recimo da se nalazimo u dnevnoj sobi i okrenuti smo ka ormaru i krevetu, ali tako da se on ne vidi u potpunosti. Odmah može biti izvršeno odsecanje grana na kojima se nalaze sobe koje nisu dnevna. Nakon toga se odseca nameštaj koji nije vidljiv, uključujući deo kreveta, recimo jastuk. Ovo može biti inicijalna optimizacija kojom odbacujemo veliki broj primitiva. Međutim zahteva dosta ručnog podešavanja kako bi ovaj sistem radio optimalno i  bio neprimetan za korisnika.



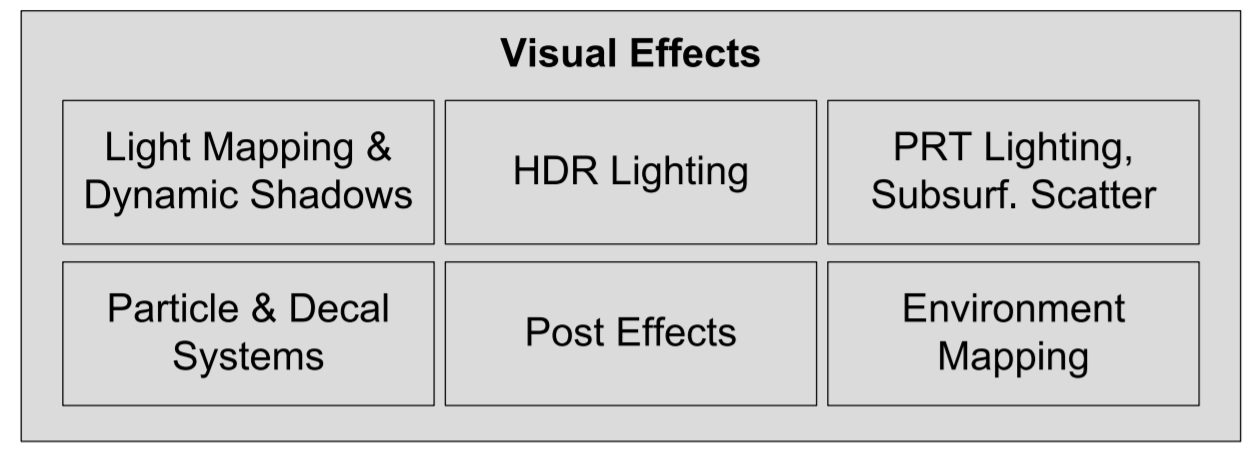
**Slika 12** Graf scene / Odbacivanje primitiva koje nisu u vidokrugu

#### Vizuelni efekti

Potrebno je obezbediti prikaz vizuelnih efekata nisu definisani kao modeli u sceni, već su predstavljeni kao poseban entitet. Služe da približe scenu realnom svetu, jer su bez njih scene samo statički beživotni modeli. U vizelne efekte se ubrajaju:

* Sistemi za čestice / particles (dim, vatra, prskanje vode, varnice...)
* Sistemi nalepnica / decal (rupe od metkova, otisci cipela, površine umazane blatom...)
* Mapiranje osvetljenosti (obračunavanje osvetljenosti modela unapred i keširanje osvetljene teksture)
* Dinamičke senke
* Post efekti preko celog ekrana (iscrtavaju se nakon iscrtavanja konačne slike scene - HDR, bloom, anti aliasing, korekcija boja...)

Kod game engine-a sistem za čestice i nalepnice je posebna komponenta koja se ostavlja korisniku na korišćenje po potrebi. Koristi se kao input low-level renderer-a. Dok se mapiranje svetlosti, dinamičke senke i post efekti nalaze ispod haube sistema za iscrtavanje na niskom nivou i iscrtavaju direktno na output buffer.



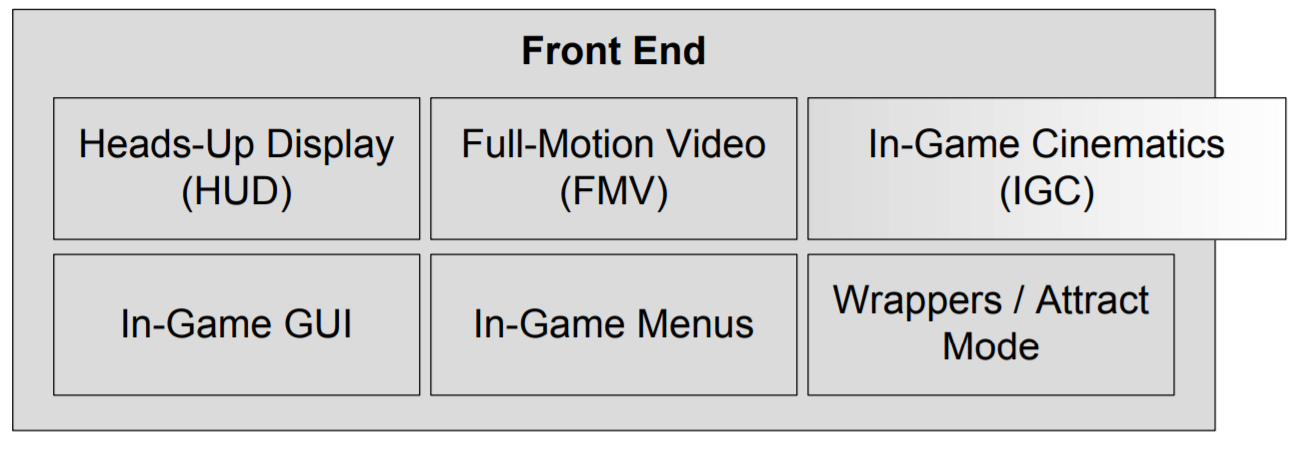
**Slika 13** Sistem za vizuelne efekte

#### Grafički interfejs

Svaka igra sadrži neki vid 2d grafike koja čini korisnički interfejs, koji služi za prikaz informacija ili kao interfejs za input. Čine ih:

* HUD / Heads-Up Display (za prikaz vitalnih informacija korisniku)
* Meni, konzola ili razvojni alati sa grafičkim prikazom

Ova grafika se sastoji od crtanja lica sa teksturama u ortografskoj projekciji (ne uzimajući u obzir udaljenost grafike). Moguće je i iscrtavanje 3d modela, ali okretati im lice uvek ka posmatraču, odnosno kameri.



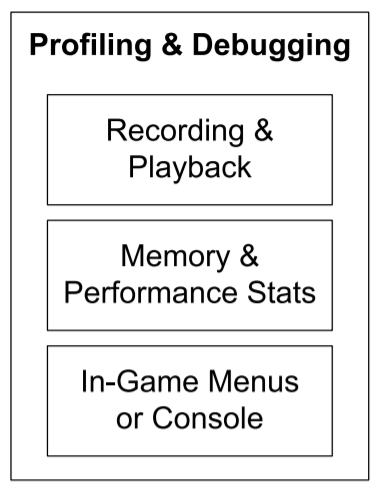
**Slika 14** Korisnički interfejs

Konzola i razvojni alati su od posebnog značaja u fazi razvoja igre, zbog toga što ubrzavaju otklanjanje grešaka, omogućavaju programeru fino podešavanje parametara u realnom vremenu, kreiranje animacija u samom engine-u, itd...

### Alati za profilisanje i otklanjanje grešaka

Od ključnog značaja za engine jeste da svaki sistem radi optimalno, zato postoje alati koji služe za profilisanje, odnosno merenje performansi. Mogu nam otkriti koje funkcije u sistemu troše više računarskih resursa nego što bi trebalo, kako bi se te funkcije optimizovale, minimizovale ili u najgorem slučaju izbacile.

Alati za otklanjanje grešaka su od posebnog značaja u fazi razvoja, jer pomažu programerima da brzo otkriju problematičan kod, a nekad će alat dati ideju i kako da se reši taj problem. Ovi alati ne ulaze u produkcionu verziju igre, ali bez njih ne bi bilo moguće razvijati igre današnjih obima.

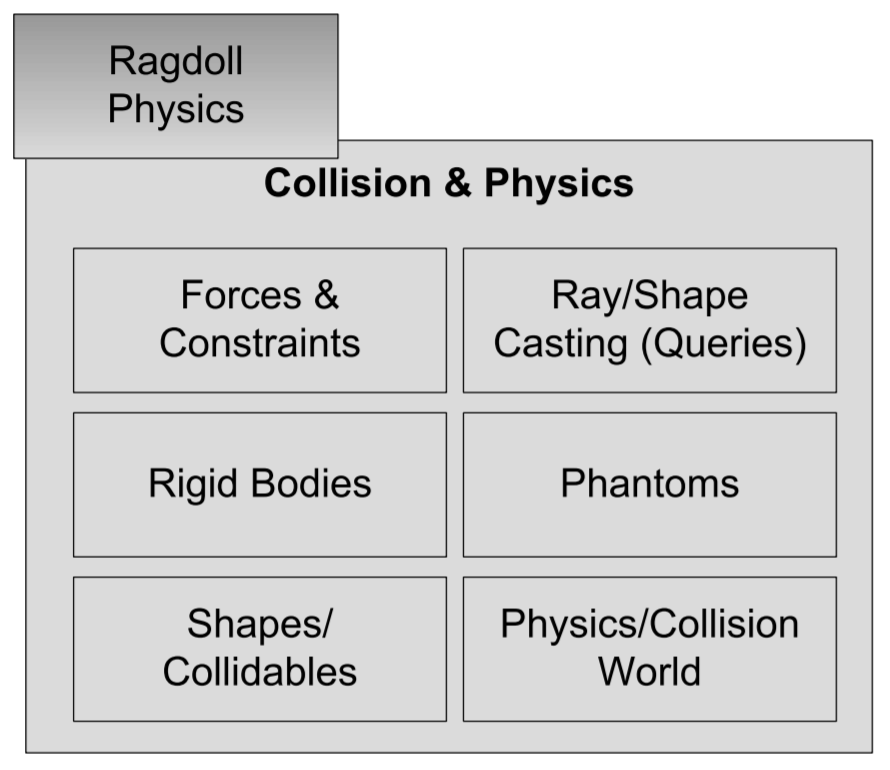


**Slika 13** Alati za profilisanje i otklanjanje grešaka

### Kolizije i fizika

Bez kolizija igre bi bile dosadne i beživotne. Ne bi bila moguća interakcija čoveka sa virtuelnim svetom na smislen način. Blisko povezan sistemu za kolizije jeste fizički sistem, tačnije dinamika krutih tela, zato što je jedino od interesa za engine kretanje tela i sile koje utiču da bi se dogodilo kretanje.

Retko koja kompanija pravi svoje rešenje za kolizije i fiziku, uglavnom se koriste opšte priznata rešenja koja predstavljaju zlatni standard za ovu vrstu sistema, kao što su Havok i NVIDIA PhysX.



**Slika 15** Sistem za kolizije i fiziku

### Animacije

Animacioni sistem je takođe komponenta bez koje ne može nijedan ozbiljan engine. Postoje više vrsta animacija koje poseduju svoje vrline i mane.

* Teksturne animacije (smenjuju se teksture na određeni vremenski period i daje iluziju animacije, nalik listanja stranica sa pokretnim slikama)
* Hijerarhijske animacije krutih tela
* Skeletne animacije
* Vertex animacije (animacije temena)

Skeletne animacije su industrijski standard za 3d igre zbog toga što omogućavaju realistične animacije tela. Naravno skelet nije bukvalno ono što nam prvo pada na pamet, već je to hijerarhijski skup povezanih delova modela, koji zajedno čine virtualni skelet za animiranje. Posao ovog sistema je da izračuna poziciju svake kosti u skeletu modela i da prosledi te informacije engine-u za iscrtavanje u vidu niza matrica transformacija. Krajnja pozicija temena zavisi od obračunate pozicije korišćenjem ovih matrica. Ovaj proces se zove “skinning”.

Nešto primitivnije rešenje jeste hijerarhijska animacija krutih tela koju je moguće realizovati korišćenjem grafa scene. Na taj sve primitive koje čine kompletan model se transliraju, rotiraju i skaliraju u odnosu na roditeljsku primitivu, i time njihova krajnja transformacija zavisi od roditelja. Ove animacije nisu idealne za živa bića gde je potrebno izobličenje samih primitiva od kojih je sastavljen model, već za čvrste objekte koji zadržavaju oblik, kao što su automobili, mašine, nameštaj, avioni, vetrenjače...

### Uređaji za ljudsku interakciju

Naravno igra ne bi bila igra da igrač ne može da ima svoj input. Igrač interaguje koristeći neki uređaj koji se ponaša kao interfejs za korisnički input, na primer miš, tastatura, džojstik, volan, itd. Precizniji naziv za ove uređaje bi bio korisnički ulazno/izlazni (I/O) uređaj. Zato što je moguće slanje povrtane informacije nazad do korisnika, na primer koristeći vibracije ili zvuka na PS5 kontroleru.

Zadatak ovog interfejsa jeste sakrivanje logike koja je vezana za specifičan uređaj i kreiranje generalnog interfejsa sa rukovanje ulazno izlaznim podacima na visokom nivou. Zadužen je za skrivanje fizičkih efekata sirovog inputa kao što su neutralizacija treperenja tastera (debounce) na tastaturi ili džojstiku, interpolacija i ublaženje analognih inputa, hvatanje momenta otpuštanja tastera itd.

### Audio sistem

Kod prezentacije game engine-a akcenat se stavlja na najnovije grafičke efekte koje nikog ne ostavljaju ravnodušnim, međutim na podjednako bitan audio sistem se najčešće ne obraća dovoljno pažnje. Iako audio sistem zvuči trivijalno jer mu je cilj reprodukcija audio sadržaja, nije baš tako naivan.

Zvuk mora da bude adaptivan, jer na njega utiču mnogi faktori koji su deo okruženja u video igri. Potrebno je modifikovati zvučne efekte kako bi zvučali organski u tom okruženju. Potrebno je obratiti pažnju na lokaciju i orijentaciju igrača u sceni, da li u prostoriji gde se nalazi postoji eho, da li postoji zvučna barijera između igrača i izvora zvuka, da li se nalazi pod vodom, i tako dalje.

Nije moguće zamisliti horor igre bez ambijentalnih zvukova, krikova i vriskova u daljini, lupanja vrata i prozora od kojih se ledi krv, ili avanturističkih igara bez zvuka zrikavca, pevanje ptica, šuškanje listova na vetru, padanje kiše i gnjecanje koje se čuje koračanjem po blatnjavom terenu.

### Mreže (Networking)

Skoro svaka moderna video igra zahteva da korisnik bude povezan na internet iz više razloga. Neki od razloga su zaštita od ilegalnih kopija, sakupljanje informacija o korisniku vezanih za odluke koje je donosio u igri, beleženje dostignuća igrača ili igra sa drugim igračima online.

Posebno je zanimljiv poslednji razlog. Online igra za više igrača predstavlja ogroman izazov za razvoj. Naime, potrebno je da se podaci šalju na najefikasniji način kako bi se smanjilo opterećenje mreže, jer podaci moraju da se šalju i primaju velikom brzinom. Pored toga mora da se proverava kakvi podaci stižu, zbog toga što neki igrači mogu da zloupotrebe nedostatke sistema, kao što je ubrzano kretanje igrača, dobijanje velike količine bogastva, izbacivanje drugih igrača, automatsko nišanjenje itd.

Slanje klasičnih http zahteva serveru za pribavljanje podataka nije optimalno rešenje, jer ima veliki overhead i mnogo je sporije od web socket tehnologija. Tačnije između igrača i servera, ili igrača i igrača uspostavlja se komunikacioni bidirekcioni kanal na kome obe strane šalje podatke gotovo konstantno. Neka od rešenja jesu:

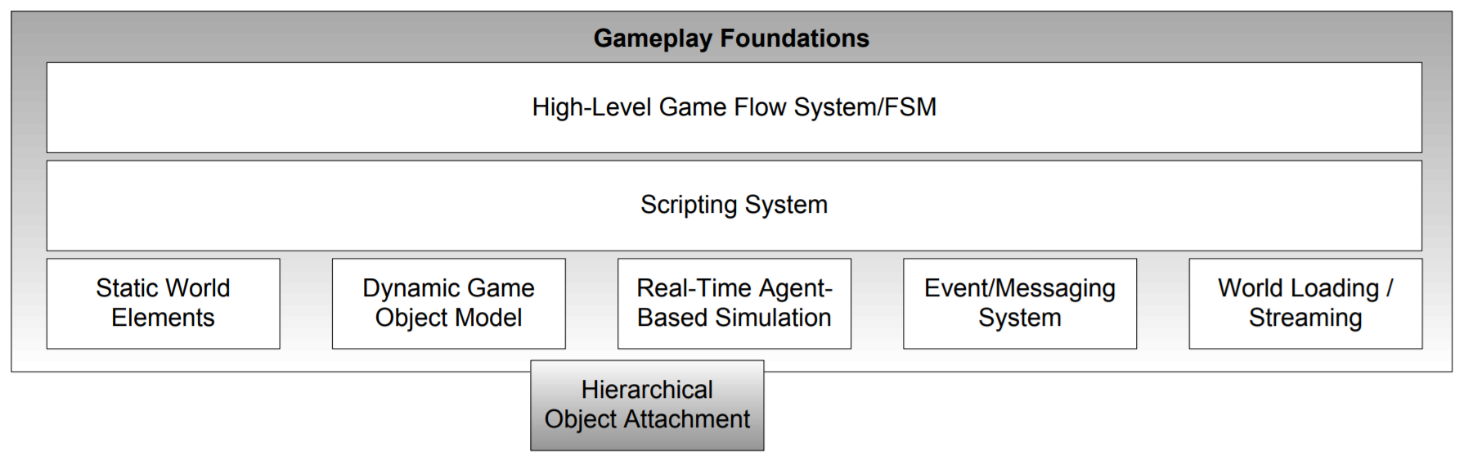
* Firebase (Guglov ekosistem koji predstavlja kompletno rešenje za networking)
* Socket.io (Hibrid http zahteva i web socketa u zavisnosti od mogućnosti)

### Osnova za igru (Gamplay Foundations)

 Pod gameplay-om podrazumevamo sve akcije koje se dešavaju u video igri, sva pravila koja upravljaju virtualnim svetom, sve mogućnosti igrača i veštačke inteligencije, i zadatke i ciljeve igrača. Gameplay elemente je potrebno odvojiti od game engine komponenti jer su oni specifični za igru, međutim postoje elementi koji mogu da se generalizuju kako bi se koristili u više projekata, kao na primer hod igrača u prvom licu, sistem za rad sa životom igrača i prikaz vitalnosti igrača, sistem za ispaljivanje metkova i određivanje šta su pogodili, itd.

Ovi elementi se obično ne pišu u jeziku u kome je pisan game engine, jer se engine-i pišu u jezicima nižeg nivoa kao što su c, c++, rust, što je dobro za performanse jezgra engine-a, ali bespotrebno komplikuje manje zahtevne skripte koje se pišu kao gameplay elementi. Zbog toga je jezik po izboru programera najčešće c#, javascript, python ili posebno pravljen jezik za taj engine kao što je Godot-ov GDScript.

Programer koji želi da pravi zabavne video igre najčešće ne poseduje znanja, a ni vremena da programira direktno nad engine-om, zbog toga je potreban sistem koji prevodi skripte i premošćuje jaz između programiranja same igre i jezgra sistema.



**Slika 16** Gameplay Foundations

Virtualni svetovi u igrama se sastoje od elemenata koji su najčešće projektovani u objektno orijentisanoj paradigmi, mada to ne mora da znači. Ovakvi elementi mogu da budu:

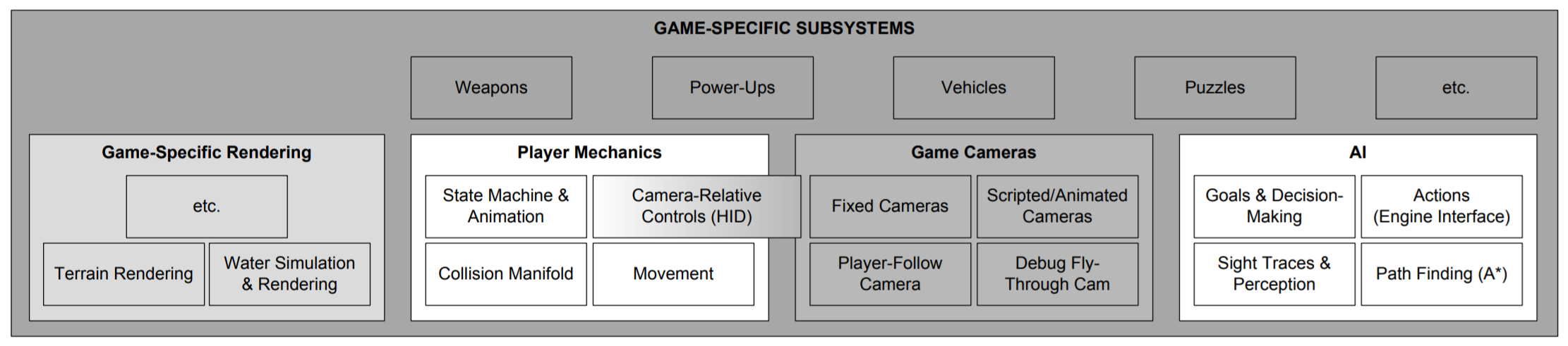
* Statička geometrija (zgrade, putevi, planine, pećine…)
* Dinamička kurta tela (stolice, tanjiri, korpe, flaše…)
* Player character / PC (kog kontroliše igrač)
* Non player characters / NPCs
* Svetla
* Oružja, odeća, vozila...

Međutim nije dovoljno samo postojanje ovih elemenata, već mora da postoji njihova međusobna komunikacija i interakcija. Zbog toga mora da postoji sistem za slanje i primanje poruka / događaja. Naravno moguće je da objekat A u sebi sadrži referencu na objekat B i C, i tako interaguje sa njima. Međutim tako se stvaraju visoko spregnuti objekti s kojima je teško raditi na duge staze. Šta ako obrišemo objekat C ili izmenimo njegovu strukturu? Potrebno je menjati objekat A, jer je zavisan od objekta C. Zbog toga je bolje poslati poruku objektu C da se dogodio neki događaj koji on može da obradi na svoj način. Tako projektovan model smanjuje spregnutost i objekat A postaje nezavisan od objekata B i C.

Implementacija veštačke inteligencije (AI) je od ključnog značaja, posebno kod igara za jednog igrača, takozvani singleplayer. Nekada se implementacija veštačke inteligencije izostavljala iz engine-a, već se ostavljalo programeru video igara da implementira za svoju potrebu. Međutim godinama su se otkrio šablon koji je deo skoro svakog AI sistema. AI sistem engine-a nije potrebno da bude preterano inteligentan, dovoljno je da implementira šablonske stvari svake inteligencije kao što su generisanje navigacionih mreža, nalaženje puteva, izbegavanje statičkih i dinamičkih objekata, komunikacija AI sistema sa animacionim sistemom i detekcija stanja i akcija igrača.

### Specifični podsistemi igara (Aplikativni sloj)

Na ovom sloju pravimo presek između engine-a i igre. Ovi podsistemi su isključivo posao za programere video igara, a ne engine-a. Naravno izuzetno je bitno da su svi niži sistemi kreirani da omoguće aplikativnom sloju zgodan interfejs za efikasan i održiv razvoj video igara. Igra može imati ogroman broj sistema u ovom sloju, sve u zavisnosti od moći aktuelnog hardvera na kome se pokreću video igre. Ovde se kreiraju komponente specifične za video igru, kao što su oružja, power ups, vozila, kamera kontroleri (slobodna kamera, kamera koja prati igrača, fiksna kamera...), kretanje, rukovanje inputom, rukovanje kolizijama, promene stanja, modela, tekstura...



**Slika 17** Aplikativni sloj

https://cs.elfak.ni.ac.rs/nastava/pluginfile.php/41045/mod\_resource/content/2/RG-P01.pdf

https://en.wikipedia.org/wiki/Computer\_graphics

https://en.wikipedia.org/wiki/Quake\_engine

Introduction to Computer Graphics, by David J. Eck

Game Engine Architecture, by Jason Gregory

1. Odnosi se na čin modifikacije harvera ili softvera kako bi se omogućile dodatne funkcije koje nisu napravljenje od strane proizvođača [↑](#footnote-ref-1)