Racunarska grafika, predavanja

Oblasti racunarske grafike:

1. **GENERATIVNA** : Generisanje grafickih objekata od jednostavnijih do kompleksnih (na vreme). Imamo svu moc (kontrolu, a samim tim i odgovornost koja s tim dolazi) nad prikazom i mozemo svaki element nekog slozenijeg grafickog prikaza oznaciti(deskriptorom najcesce) i labelisati (tj. cuvati informacije) . Na taj nacin mozemo izvlaciti informacije iz nekog graf. prikaza (pretrazivati). Radi u realnom vremenu.
   primer : Guglanje, iz tekstualnog polja za unos klikom na dugme dobijamo graficki prikaz rezultata pretrage.
   Pretraga informacija je pomocu graf. prikaza preciznija i jednostavnija.
   Pretraga "zene sa crvenim salom" dobijamo slike zena sa crvenim salom razlicitih rasa, boja kose, ociju, godista, u fokusu ili u pozadini.
   Problem je sto ne mozemo SVE da prikazemo i zato je nekad bolja slika(video)

2. **OBRADA I ANALIZA SLIKE**: Slika je uzorak realnosti (digitalizovana realnost) koja dolazi u racunarski sistem i obradjuje se. Slika se obradjuje iz 2 razloga:

   - Radi istine (da racunar izvuce neku informaciju tj, prepozna nesto) : Rasparcava se i postaje neprepoznatljiva radi izvlacenja informacija ili prepoznavanja (npr patterna). Nakon ove obrade slika dolazi do neprepoznatljivosti, nije ni poenta da ostane "lepa coveku na oko", jer treba da bude transformisana u oblik takav da bude prilagodjena algoritmima koji vrse analizu.
   Zanimljiva je za video nadzor (iz bezbednosnih razloga).

   - Radi lepote : Matricne transformacije nad pikselima slike (primer : Photoshop). Problem lezi u tome da se prilikom obrade popravlja nesto sto je lose na slici, ali se moze pokvariti nesto sto je bilo dobro.
   Koristi usluge generativne grafike, jer se oko predmeta obrade najcesce iscrtava neki okvir.

   Obe stvari se odnose I na staticke slike I na uzorke videa.
   DSP (digital signal processing)

3. **KOGNITIVNA GRAFIKA**: Koristeci 1. I 2. uspostavlja relacije I zvodi zakljucke!
   primer : Obrada/analiza medicinskog snimka u potrazi za nekom bolesti, na snimku se oznacava region koji moze ukazati da nesto nije kako treba.

Osnove grafike, kako je grafika organizovana?

Graficki pipeline: Resavanje prvo osnovnih problema uz pripremu da onaj koji dolazi naredni moze to da prihvati, unese vise detalja I pripremi to sto salje za narednog u pipeline-u sve dok se ne pripremi graficki prikaz takav da svako ko gleda u njega moze da donese isti zakljucak o tome u sta gleda (bar osnovnim pojmovima). Moguce ga je paralelizovati.

1. **Tacka**:
   Osnovni gradivni element. Sama pod sebe nam ne moze pomoci jer je tako jednostavna.
2. **Linija** :
   I dalje osnovna. Poveziujemo tacke, ali izmedju mnogo tacaka postoji mnogo mogucnosti povezivanja. Povezivanjem dobijamo ZICNI MODEL. On nam moze posluziti da vidimo da li su dobro postavljeni odnosi. Blizi smo sceni, ali i dalje ne vidimo sta je pokriveno, a sta suplje. Znamo da postoji odnos izmedju obekata ali ne znamo kakav!

3. **Trougao** :
   Planarni (u jednoj ravni) oblik, povrsina prikazana pomocu 3 tacke (najjednostavniji je). Trouglom se predstavlja bilo koji poligon koji postoji. Lako se matematicki objasnjava. Teme trougla -> Verteks. Ivica trougla -> edge
   Hardver koji radi sa trouglovima -> graficki procesor!
   Uvodimo trouglove da bismo mogli dobiti informaciju o povrsini objekta.

4. **Transformacije** :
   Uzastopne transformacije kako bi odnosi izmedju svih trouglova na sceni ostali isti (npr pri prikazu kretanja) ili prilikom pretvaranja postojeceg oblika u drugi (imamo model konja, a treba nam kamila).Sprovode se nad temenima (a samim tim i nad citavim objektom)!

5. **3D/2D Projekcija**:
   Projekcija 3D sveta na 2D ravan pomocu nekog uredjaja koji ima svoju poziciju u prostoru (npr kamera) i to se projektuje na prikazni uredjaj. Najvise posla uzima.

   Skrivanje: Sakrijemo (ne brisemo) sva temena viska (ne prosledjujemo ih dalje na obradu).

6. **Clipping** :
   Sada se mogu sakriti (nikako izbrisati!) sva temena viska (koja ne idu na dalju obradu). Veoma zavisi od prikaznog uredjaja (velicine ekrana). Redukcija objekata sa scene samo na”ono sto se vidi”. Ovde se mozemo posvetiti detaljima (npr boja, sta je sakriveno/otkriveno itd...)
   Cesto ide zajedno sa projekcijom jer se nakon clippinga dobija deo scene koji se zaista vidi.
   Treba da isecemo onaj deo koji ce zaprvo da se vidi (donosi dodatne probleme jer se iskljucivo radi algoritamski)

7. **Bojenje** :
   Bojenje != Boja fragmenta.
   Objekte i njihove fragmente uvek gledamo pod barem jednim izvorom svetlosti (ili vise). Svetlost uvek pada na neku povrsinu I od ugla I intenziteta svetlosti zavisi boja svakog elementa i tako prepoznajemo oblike. Zato uz bojenje uvek ide i sencenje! (navikli smo na takav pogled)
   Iz HCI-ja boja privlaci paznju ali je i ne zadrzava

8. **Sencenje** :
   Iluminacija != sencenje (tehnika bojenja) != shading (bacanje senke)
   Zavisi od udaljenosti, osvetljenosti i polozaja objekta posmatranja.

9. **Teksture** :
   Prevlaka (pakovanje) u vidu 2D slike. Mehanizam jos vece realnosti. Navlacenje 2D povrsine preko nekog 3Doblika, ali bez guzvanja! (primer : rucno pakovanje lopte u ukrasni papir vs dinosaurus sa krznom zeca)

10. **Spajanje** :
    Refleksija, zracenje za stola povecati i promeniti znak (+, -) y koordinate objekta.

*Greska na slici (statickoj) moze proci neprimeceno, dok losa animacija pokreta ne moze ostati neprimecena.

Kako kocku prebaciti u 2D svet?

Prebacimo ono sto kamera vidi u 2D svet i to onda mapiramo na 2D svet ekrana TV-a, projektora, itd. (tj. sta god aktivna kamera vidi ce biti projektovano).

Rasterizacija - punjenje frejm bafera pikselima grafickog prikaza (tj. kada se napravi projekcija na 2D ravan onda treba da se to pripremi za prikaz)

Osnovna grafika (koju mi radimo) zasnovana na transformacijama pomocu matrica.

Ray-tracing grafika - mnogo mocnija i realnija.

Podela grafike po prirodi (u pogledu vremena)

1. **REAL-TIME**
2. **NON REAL-TIME**

Real-time :

Trazenija. Graficki prikaz (rezultat izvrsavanja grafickog softvera) se generise u unapred odredjenom vremenskom intervalu. Soft (sme da stuca) vs Hard(ne sme da stuca) real-time. Obicno je vreme generisanja odgovora jednako vremenu akcije korisnika (primer: kretanje lika u igrici pritiskom na strelicu, mora se pomeriti odmah). Igrice npr koriste soft : mozes malo da zakazes, ali nemoj cesto I nemoj previse, dok se u automatici koristi hard : odziv ne sme da porani niti da zakasni.

Primeri primene : razlicite simulacije (letenje avionom za buduce pilote), virtuelne operacije, simulacija prirodnih katastrofa, udar bombe...

Simulacije predstavljaju prvu ozbiljnu upotrebu racunarske grafike. Nekada (npr. za letenje u avionu) zahtevaju imersivnost tj. da ne postoji osecaj kao da se vrsi simulacija

Non real-time :

Kod rezultata nije naglasak na vremenu, vec na kvalitetu! Najbolji primer su filmovi, gde je potrebno precizno i detaljno generisati sadrzaj jer ce se koristiti (konzumirati) mnogo kasnije od momenta generisanja. Cesto se odredjene scene/elementi generisu po nekoliko dana, ali je rezultat daleko precizniji (renderovati pomeranje glave sa 60 000 vlasi kose).

*Pitanje : Koja je bolja/Kako spustiti cenu real-time grafike?
Odgovor : Neke danasnje real-time tehnologije dostigle kvalitet non real-time (Nvidea, renderovanje muske koze), ali i dalje posoji problem u dinamici (sve je ok dok lik stoji mirno, ali daj mu instrukciju da namigne). I dalje real-time nije otporan na kompleksne animacije!

CAD, CAA, CAM

- **Computer Aided Designe**: Resio problem crtanja modela na papiru, predugo otklanjanje gresaka(javljanje greske bi uslovilo da se sve ponovo mora crtati!), i omogucio ponovno koriscenje vec napravljenih stvari iz starih projekata u novim projektima. Alat takodje omogucuje generisanje i renderivanje(2D/3D) prikaza skice, znacajno ubrzava rad!
  Najcesca primena u arhitekturi, masinstvu, gradjevini...
- **Computer Aided Asembly**: Koristi izlaze CAD elemenata i daje preporuku za proizvodnu liniju (kako najbolje sastaviti proizvodnu liniju uz resurse).
- **Computer Aided Manufacture**: koristi izlaze CAA i racuna koliko cega treba za sta tacno i kako najpovoljnije isporuciti kome je potrebno u najmanje vremena.

Vizuelizacija

Prikaz toliko podataka i informacija koje je potrebno prikazati. Prezentacija najcesce (barem)desetinu hiljada necega i vise. (Sada na manjim prikaznim uredjajima sto je dodatni problem jer vazni detalji moraju biti vidljivi i na njima)

*Primer 1 : Medicina, za vreme operacije ako se nesto ugradjuje u ljudsko telo (na neko mesto, pod nekim uglom) kolika ce biti pokretljivost zgloba, bojenje krvnih sudova...

*Primer 2 : Vizuelizacija konekcije informacionih sistema i priroda veze (trag koji svako a internetu ostavlja).

*Primer 3 : Vizuelizacija govornih (audio) signala u cilju otklanjanja suma.

Virtuelna i augmentovana stvarnost

1. **Virtuelna** : Sva korisnikova cula su vestacki nadrazena.
2. **Augmentovana** : Samo neka korisnikova cula su vestacki nadrazena u cilju prosienja stvarnosti.

*Primer/pitanje : Zaposlio si se, uzimas projekat, da li ces izabrati augmentovanu ili virtuelnu? Odgovor : Augmentovana je zahtevnija, covek vestacki aspekt ne sme prepoznati! To je teze modelovati, programirati… Takodje i hardverski je zahtevnije zbog opreme koja to treba za prikaze i/ili obradi i zbog zahtevne procesne moci. Zvuk je veoma tesko “lagati”.

Rezmi rada

Programeri koriste graficke biblioteke za pravljenje grafickih prikaza da transformisu zahteve visokog nivoa u niskonivovske zahteve grafiskog hardvera.

Postoje 2 rezima rada:

1. **IMMEDIETE-MODE**:
   Neposredni rezim.
   Iscrtava se trenutak po trenutak. Mi definisemo svaki frejm! (Punimo video memoriju graficke kartice frejmovima koji ce biti prikazani i ako nesto izostavimo nece biti nacrtano, tj imamo kontrolu ali i odgovornost) Saobracaj izmedju graficke kartice i podakata je vaoma zahtevan, ali mi imamo potpunu kontrolu nad njim. Programiranje iskljucivo proceduralno, preferira se strukturirano (ne objektno) zbog brzine. Sve mora biti sinhronizovano sa iscrtavnjem tj. mora biti citav frejm iscrtan pre nego sto krene iscrtavanje narednog. Prikaz od 15 slika u sekundi covak shvata kao pokret, 50-60 ako zelimo prikaz bez reckanja, a 100 samo za stvarno velike ekrane.

2. **RETAINED-MODE**:
   Posredni rezim, vise je prisutan.
   Iscrtavanje frejma u frejm. Biblioteka to radi za nas. Punimo radnu memoriju podacima vezanim za scenu I ona ce prikazivati te frejmove/podatke brzinom koju smo naveli da nam treba. Mi ne vrsimo sinhronizaciju sa prikazom, to radi sada biblioteka. Mi punimo u opisu samo promene scene (onda se scena azurira). Saobracaj podataka je manji. Za programiranje se koriste deklarativni jezici. Problem je sto imamo onoliko slobode za opis scene koliko nam biblioteka koju koristimo dozvoljava (diktira). *OPENGL omogucava da se spustimo malo nize npr zbog blurovanja (tj naglasavanja), to je taj nesinhronizovan prikaz

Vektorska i rasterska grafika

Vektorska : Iscrtavanje liniju po liniju, povecavanje odnosno smanjivanje ne utice na kvalitet prikaza. Graficki prikaz cini skup vektora.

Rasterska : Iscrtava se tacka po tacka, od gore do dole, s leva na desno (logicki px po logicki px).

*Piksel(Pictute element) se moze posmatrati kao **hardverski**(na ekranu) ili **logicki**(opis slike frame buffer-u, pravi se u programu, ne mora se prikazati).

*Brzina osvezavanja : Brzina iscravanja jedne slike (frejma) na ekranu. Posto se rasterska iscrvata piksel po piksel, treperenje prilikom osvezavanja je vece. Zato je vektorska prijatnija za oko.

*Glavni nedostatak vektorske : Ne moze detaljno i nijansirano prikazati unutrasnjost objekta! Bukvalno bi se linija po linija povlacila unutar objekta, sto bi rezultovalo u istovremenom iscrtavanju ooogromnog broja vektora cija duzina najcesce ne prelazi nekoliko px. Nastao bi veliki problem prilikom osvezavanja, znacajno smanjene performanse. Zavisi od max broja vektora koji je moguce istovremeno iscrtavati.

Rasterska ionako uvek sve crta, pa nije problem da se detaljno boji unutrasnjost objekta. (tj i onako vec prelazi preko unutrasnjosti samo je ne boji, a ako treba da se boji vreme za koje to uradi je zanemarljivo malo). Fotoaparat je rasterski uredjaj (pa je zato on "pogurao" rastersku grafiku)

*Pitanje : Prodavnica : kako kupiti vektorski/rasterski sistem?

- Za **Vektorski** bitno : **broj boja** i **broj vektora**(sto vise to je mocniji, al i skulji) koje prikazuje! Rezolucija i velicina ekrana nisu bitni.
- Za **Rasterski** bitno : **broj osvezavanja** (oko 60 sto znaci da za 1/60 sekunda mora da se predje preko svih piksela i da se osvezi to sto treba), **rezolucija** (broj px po visini i po sirini) i **velicina ekrana** i **broj boja**. Bilo bi dobro uracunati i razdaljinu gledanja (ne gleda se TV sa iste rzdaljine kao mobilni, prema tome velicina drugacija).

*Pitanje : Da li bi vektorski sistem mogao prikazati tipicnu fotografiju (npr sa letovanja)?
Odgovor : Da, mogao bi, ali bi najverovatnije bio potreban po 1 vektor za svaki piksel! (vektori simuliraju rasterski prikaz, u tacki u kojoj pocne vektor tu se i zavrsava) Postoji hardversko ogranicenje ako je broj px slike > broja vektora koje sistem moze prikazati! Nije optimalno.

*Vektorsko opisivanje na rasterskom prikazu je sasvim normalno, racunar (softver) uvek radi vektorski. Blize nam je zbog naseg obrazovanja (ucimo matematiku od osnovne skole)

Softverski nivoi rastera i vektora

Nivo Komande: Da li softver sa korisnikom prica pomocu rastera ili vektora tj da li korisnik daje rasterske ili vektorske naredbe. Vektorski ili Rasterski, koji nam je blizi?
Nama softver daje vektorske naredbe. Stvari zapisuje matricno i transformacije se predstavljaju matematickim transformacijama nad matricom.
Primeri vek. softvera na nivou komande: Word, Corel draw, CAD sistemi (tj ozbiljni inzinjerski softveri)…
Primeri ras. softvera na nivou komande: Paint, Photoshop…

Nivo Podataka: Rasterski ili Vektorski
Odnosi se na to kako se taj graficki prikaz memorise

Rasterski formati:

- .tiff : skeneri, ucitava po tackicama boju i zapisuje
- .jpeg : kompresona tehnika vidu furijeovih transformacija(kosinusne). Izbacuje najfrekventnije i zavrzava manje frekventnije. Sto vise frekvencija promena ima fajl je vaci. Ovo je rasterski format ali sa kompresnom tehnikom.
- .raw : najveca tacnost, ima jaaako puno podataka, svaki proizvodjac fotoaparata ima svoj sirovi rast. zapis
- .png : prenosiv, namenjen za mrezni premos
- .bmp : px po px zapisan u sekvencijalnom fajlu (px po kolonama svakog reda, takozvana bitmapa). Moze ga prikazati vektorski sistem isto kao i sliku.

Vektorski formati :

- .cdr : zapisivanje pomoc vektorskih komandi, nativni forat za Corel. Moze ga prikazati rasterski sistem ali se mora rasterizovati.

*Primer: Crtanje cica glise (slika):
1. Rasterski : mapa koja belezi NxM piksela, ne zavisi od detalja
2. Vektorski : 6 vektorskih komandi, zavisi od detalja

*Pitanje: Sta je .wmf? Windows Meta Files, kombinacija je rasterskog I vektorskog, direktna je slika Microsoftovog grafickog engine-a. Primer takvog formata je .pdf. Prednost je sto na Windows masini ce prikaz ovog formata biti najbrzi moguci, a mana je sto prikaz npr na Unix-u zahteva konverziju u ekvivalentni format, sto uzrokuje gubitak na brzini!

Fizicki hardver (fizicki nivo rastera i vektora)

Vektorski i rasterski, u oba slucaja prisutan CPU. Ako imamo samo CPU, takav hardver je pogodan za jednostavnije graficke prikaze (npr fiskalna kasa), dok u ostalim slicajevima potreban nam je I GPU.
Prema tome rasterski hardver moze biti organizovan kao:

1. **bez GPU**(sa deljenom ili izolovanom memorijom)

2. **sa GPU**(sa deljenom ili izolovanom memorijom)

U oba slucaja pristupa se operativnoj memoriji koju koristi CPU, samo ukoliko smo investirali u GPU (pogotovo ako je mocan) nema smisla da on mora da pristupa njemu sporoj operativnoj memoriji sa niskim nivoom paralelizma, tako da je komunikacija spora jer se prenose ogromne kolicine podataka.
GPU sa sistemskom memorijom prakticno slabo vredi.
Zato se koristi **memorijski izolovana graficka memorija**, Ona se proizvodi drugom tehnologijom, da bi na jednoj procesorskoj jedinici bilo mnogo kapaciteta, uz veliko osvezavanje I veliku brzinu pristupa u velikom paralelizmu. Njoj pristupa posebnim magistralama uglavnom iskljucivo graficki procesor. Tada u sistemu egzistira I operativna I izolovana graficka memorija.

*Primer/Pitanje: Iscrtavanje cicaglise. Zasto koristimo IV kvadrant prilikom opisivanja onoga sto crtamo?
Kako bismo se prilagodili masini, tj. da se ne pravi razlika izmedju ispaljivanja snopa za iscrtavanje I opisa onoga sto se iscrtava I da se koordinate opisa mogu “as is” prebaci u sam hardver prikaznog uredjaja!

***Vektorski HW (uvek ima GPU), tok aktivnosti:**

Opis periferije (najcesce unet pomocu tastature) ne ide direktno u CPU da se ne bi non stop izazivali prekidi I da se njegov rad ne bi usporavao. Zato se opis cuva u memoriji.Kada zahtev za prikazivanje/iscrtavanje bude upucen CPU pristupa se tom opisu, ali njemu ne pristuva CPU. Kada takav zahtev stigne, opis se prevodi u vektorski zapis koji je prepoznatljiv video procesoru(na slici, inace GPU). GPU prihvata od CPU taj opis, ne direktno, vec ucitava iz memorije I pocinje da interpretira to I iscrtava odredjenim naredbama dok mu CPU ne javi da je doslo do promene I da treba da ucita novi opis.

***Rasterski HW bez GPU, tok aktivnosti:**

1. Sa **deljenom memorijom**:
Video kontroler nije procesor vec nisko inteligenta periferija, obicno vrsi privremeno memorisanje ili prebacuje zapisa iz jednog oblika u drugi. Koristi sistemsku memoriju od koje je pozajmio/rezervisao jedan deo. Najcesci je u sistemima koji ne zahtevaju posebne graficke prikaze.

2. Sa **izolovanom memorijom**:
Pojavljuje se posebna, video memorija (Jer su korisnici hteli vise, tj mocnije graficke prikaze). U praksi malo prisutan. Cemu ulagati u posebnu video memoriju ako nemamo graficki procesor?
Monitor ne zna da ispisuje vektore, vec prikazuje px redom od gore do dole, s leva u desno. Na ulaz mu dajemo vektorski opis (pomocu neke periferije) koji je neophodno rasterizovati kako ga je moguce prikazati. Opis onoga sto se iscrtava ide u memoriju. Posto nema GPU, taj opis CPU procita I rasterijuze! Vektorski opis pretvara u opis “tacka po tacka” I taj opis se prebacuje u video memoriju.
Sada do naredne naredbe za promenu scene prethodno opisan proces se ne ponavlja, vec video kontroler cita frejm bafer(u njemu je onaj opis tackicu po tackicu) I tako naredjuje monitoru da opali ili ne opali taj px. Kada

opali poslednji px, nastupa period osvezavanja I to se vrti tj. mora video kontroler da se sinhronizuje s osvezavanjem i opet da krene da cita opis prvog piksela i tako u krug! Ako je osvezavanje npr 60Hz onda ce video kontroler ovaj zapis da cita 60 puta u sekundi.

*Frejm bafer : deo video memorije u kom se cuva opis jedne slicice koja se prikazuje. Njegova pojava je bila pocetni korak ka organizaciji memorije u celine tj. bafere, tako da svaki bafer cuva 1 frejm. Ovo je primer ubrzavanja grafike bez hardverskih integracija.

*Primer : Animacija bele macke, koja se sastoji od N frejmova. Korisnik gledajuci frejm za frejmom dobija utisak kretanja te macke. Krecemo sa prikazom. Sistem prvo ceka da mi napunimo prvi frejm programskim opisom koji opisuje prvu slicicu. Kontroler potom krece da ga prikazuje na monitoru. Dok on to radi, procesor puni naredni frejm i onda se u jednom momentu radi switch tj. ono sto je bilo front postane back, a onda se opet sprema novi back .Tako sve dok se ne napuni I ne iscrta svih N frejmova.
OpenGL npr ima 4 frejm bafera u primitivnom obliku, zovu se Front I Back i Left i Right (za levo i desno oko za 3D prikaze).

***Rasterski HW sa GPU, tok aktivnosti:**

1. Sa **deljenom memorijom**
   Koristi memoriju preko magistrale, ali su proizvodjaci maticnih ploca odvajali jednu brzu magistralu, cisto zarad performansi.

2. Sa **izolovanom memorijom**
   Komunicira sa sopstvenom memorijom sa kojom komunicira sa svojom magistralom.
   Unese se vektorski opis grafickog prikaza. On ode u RAM memoriju, odatle ide u video procesor. Tu se vektorski opis transformise u rasterski. To je dosta lep proces za paralelizaciju. On puni frejm bafer, pa video kontroler to prikazuje brzinom osvezavanja.
   Svaki GPU moze da prikazuje od nekoliko desetina do 100-120 promena u sekundi.

Video kontoler klasicne arhitekture:

Osnovni zadatak mu je da **cita opis** prikaza iz video buffera/frejm bafera/video memorije I naredjuje prikaznom uredjaju da ga prikazuje (monitor, ekran, projektor…).

Primer sa slajda(17): 1 lokacija frejm bafera je zaduzena za opis boje 1 px. Postoji veza izmedju adrese lokacije(redni broj lokacije) I koordinata piksela koji se prikazuje. Ukoliko imamo sliku NxM, prvih M lokacija frejm bafera opisivace prvi red prikaza, a svakih narednih M naredni red I tako N puta.
Sto je imamo vise bita u memorijskoj lokaciji, u stanju smo da prikazemo vise razlicitih boja! U primeru svaka lokacija ima 9 bita gde se svaka 3 bita odnose na po jednu komponentu boje (RGB model boja).
Inace racunari rade sa true color sistemom, tj. po 8 bita za svaku komponentu boje, tj 24 bita.

Opis rada: Frejm bafer drzi opis. Taj opis se treba naci na ekranu. Video kontroler je sastavljen od “ILI” logickih kola. U zavisnosti od ulaza na izlazu daje analogni signal (valjda 8 puta veci?) koji nastaje tako sto odradimo logicko ili ulaznih signala. Prema tome, sto veci binarni zapis dobili na ulazu, visi signal ce se naci na izlazu. To se odrazava I na komponente boje. Npr ako je signal crvene velik, plave srednje, a zelene ni nema, px ce biti obojen u ljubicasto.

Scan kontroler: Prati sta ce monitor sledece da prikaze, tj nad kojim ce px snop za ispaljivanje biti u sledecem trenutku I da se ocita adresu u memoriji koja je za njega namenjena. On sinhronizuje ucitavanje memorije sa trenutnom pozicijom

prikaznog sistema monitora.

*Problem: Tehnicki je korektno, ali ima nedostatak. Malo su se razbacivali. Hardverski je moguce prikazati sliku sa 2 milijarde boja, ali se to u praksi nikada ne desava. Takodje, ukoliko pokusavamo da prikazemo istu boju vise puta na slici ona se svaki put mora zapisivati u frejm bafer. Ovo resenje nece stucati, ali rasipa resurse I puno kosta.
*Resenje: Ne cuvamo opise svih px u video memoriji "as is". Potrebno je formirati hash tabelu u koju se smesta egzaktni broj boja prisutnih na slici, a u video memoriji se samo indeksiraju konkretne boje.

LUT arhitekture:

Svaka lokacija video memorije treba da ima onoliko bitova koliko je paleta boja dugacka. Lokacija ne opisuje boju px vec adresira tu boju u paleti boja, a u LUT se nalaze sve boje koje postoje na slici. Indirektna je verzija, lookup tabela je neka vrsta posrednika (middleman). I dalje postoji scan kontroler koji obavlja sinhronizaciju. Jeftinije je resenje, sto je vise boja koje zelimo da prikazemo to je usteda veca (sa aspekta memorije) jer ne cuvamo sve moguce boje vec samo one koje su na slici.

*Pitanje1: Kako ide prikaz kod standardnog I kod ovog sada, primer prikaz "smajlila"?
Kod klasicnog opisi boja stoje u memoriji tj u frejm baferu. Kod LUT-a bi u celoj lookup tabeli bile 3 boje. A u citavoj memoriji frejm bafera bi se nalazila jedna od tri adrese boje. Ako bi svaka lokacija frejm bafera imala 9 raspolozivih bita, moglo bi se adresirati 2^9 = 512 boja.

*Pitanje2: Sta cemo ako dodje slika sa vise, npr 700 boja? Zahteva ponovnu alokaciju lookup tabele, a samim tim I frejm bafera da moze da prikaze vise boja. Zahtev za vecim brojem boja ce uneti malo stucanje dok se ova ponovna alokacija I punjenje ne zavrse.

*Pitanje 3: Prikaz .bmp slike? Svaka lokacija opisuje boju piksela u slikama .bmp formata. Prema tome, moramo sekvencijalno proci kroz celu sliku I kada naidjemo na novu boju, dopisemo je. Moramo ponovo proci kroz celu sliku, procitamo boju, najdemo njen indeks u lookup tabeli I njenim indeksom napunimo lokaciju u frejm baferu video kontrolera.

*Pitanje 4: Koji rasterski format ne zahteva predprocesiranje ukoliko koristimo LUT? Odgovor: **.gif**
Njegov format je takav da ima prvo neko zaglavlje, pa ide paleta boja, pa opis px koji jeste indeks boje.

*Pitanje 5 : Vojna letelica, kakav sistem u nju ugraditi? Obavezno bez indirekcije("middle man-a")! Brze je, ali brzinu placaju veci memorijski zahtevi.

Pipeline

Proces proizvodnje od vektorskog opisa geometrije do krajnjeg rasterskog prikaza na ekranu. Objasnjava se serijski/linijski, iako se neki njegovi delovi mogu paralelizovati/mesati. Svaki deo pipelina radi svoj deo posla koje je malo zrno ukupnog problema.Delovi:

1. Aplikacija — negraficki deo
2. Geometrija } graficki deo
3. Rasterizacija }

***Aplikacija** :
Glavna je. Radi skoro sve, osim grafickog dela (komunikacija sa korisnikom, tj.rukuje ulazima korisnika , ocitava ih i interpretira, izracunavanja, prati promene u poziciji i salje grafickom delu za prikaz...).

Primer: Igrica. Imamo nekakvu aplikativnu i/ili poslovnu logiku. Tj. to su podaci I kod koji nose ostalu funkciju. Njih uvek izvrsava CPU. Za to izvrsavanje se koriste operativna memoija I spoljna memorija. Aplikacija je zaduzena za svu komunikaciju sa korisnikom. Ona ne obavlja prikazivanje grafickih elemenata, vec proracunava geometriju(npr lika u igrici), fiziku kretanja (Postoje i graficki procesori koji rade fiziku i kretanja ali oni su dosta skupi) itd.i salje grafickom delu na prikaz.Takodje radi evidenciju bodova, primenu strategije, multiplayer komunikacija… Moze da radi neke graficke stvari (npr. neke promene da izracuna i posalje grafickom delu da iscrta, primer za to je udarac u igrici pa da se sracuna promena oblika (npr. lica) pri tom udarcu)
Glavni zadatak aplikacije je da grafickom delu prebaci sve **primitive**(sa bojom, sencenjem, itd)svih**objekta i karaktera (krecu se svojom voljom) sa 3D scene** koja treba da se prikaze! Moze to da radi za svaki frejm (neposredni/immidiete rezim), a ako se prvi put cita opis a posle samo izmene onda smo u posrednom/retained rezimu.
Na sceni se osim opisa objekata moraju nalaziti I opis gde su I u koju su stranu usmereni **svetla**(esencijalno, bar ambijentalno, bez njega se nista ne vidi, a tackasta ne moraju da postoje i ona nam iskazuju neke osobine na objektima) I **kamere** (da znamo tacku iz koje se posmatra).

*Zadatak grafickog dela: Prikaz 3D sveta na 2D ravni (uredjajima) sto vernije!
Sta treba poslati grafickom delu za prikaz? Opise:

- **Objekata** : pasivni su na grafickoj sceni jeste nesto sto je postavljeno na nju I nema svoju volju
- **Karaktera** : aktivni su, imaju svoju volju i interaguju sa drugim karakterima
- **Dodatne informacije** : npr boje, teksture, izvori svetla, kamere (moraju imati poziciju, oblik, gde gledaju, tip, koliko je sirok objektiv, itd.) , itd...

- Koraci I zasto je zasto je organizovano u pipeline?

Pipeline: logickim redosledom logicki procesi koji se moraju izvrsiti.
Seti se primera sa Springfieldom I Simpsonovima!
Svaki deo/faza pipeline-a radi svoj deo posla u velikom zadatku, koji kada ga resi predaje sledecem u pipeline-u. Prvo se zato igramo sa osnovnom geometrijomtj. sa **temenima**(samim tim I linijama, trouglovima). Kada se sve nad temenima odradi sto je bilo predvidjeno, dolazimo na **projekciju I skrivanje**(uglavnom idu zajedno u paraleli).Tu se dalje na obradu prosledju je samo temena trouglova koji upadaju u view kamere projektovane na 2D ravan. Pri toj projekciji se vrsi I skrivanje koji izbacuje sve one podatke koji se ne vide iz ugla kamere. Nista se ne brise iz opisa scene, vec se samo ne prosledjuju informacije o delovima koji se ne prikazuju (daleko manje podataka se salje na obradu). Do sada smo govorili samo o temenima tj. kada se izvrsi projekcija na 2D ravan zavrsavamo sa geometrijom I otpocinje rasterizacija. **Proces rasterizacije** obuhvata pronalazenje unutar svakog trougla (definisanog sa 3 temena) sve elemente(fragmente) koji je usao u view kamere, koji ce se kasnije pretvoriti u piksele! Njih pronalazi tako sto zakljucuje kako sta treba **obojiti I osenciti** postujuci uglove I prirodu izvora svetala.Nakon toga se lepi **tekstura I vrsi spajanje**(“mergeovanje”). Izlaz iz pipeline je 2D slika tog pogleda 3D scene koja treba da se prikaze!

*Tekstura : Prevlacenje 2D slike preko 3D objekta, bez potrebe za izmenom u geometriji tog objekta. Obezbedjuje verniji prikaz sa manje posla. Ako koristimo teksture da bismo vernije docarali materijal npr ne moramo da se mucimo i stravamo haoticno mnogo poligona kako bismo docarali materijal, vec 2D slikom oblacimo 3D objekat.

*Ambijentno svetlo: 1 svetlo koje nam obezbedjuje da ono sto se nalazi na sceni zapravo mozemo videti (nema ni jednog svetla, ni mi nista ne vidimo). Ono ima svoju poziciju i intenzitet. Na sceni moze pored njega biti proizvoljno mnogo drugih tackastih izvora svetlosti, sa svojim intenzitetom i pozicijom.

*Tackasti izvor: Daje dovoljno skoncentrisane svetlosti da nase oko bolje vidi oblike, boje, matrijale itd…

*Kamera : Oko(oci) korisnika! Ona odredjuje koji deo 3D sveta cemo projektovati na 2D ravan. Ima svoje koordinate i polozaj. Moze na sceni biti postavljeno i vise kamera (npr multiplayer, detaljan prikaz objekta iz vise uglova...). Kamera se **moze po potrebi iskljuciti**, kod svetla toga nema, ako je na sceni, svetlo se prikazuje.

*Projekcija i skrivanje : mogu ici zajedno (u paraleli/ o jednom trosku). Konzumiraju najvise procesorskog vremena u

celom pipeline. Razliciti korisnici (kamere) mogu gledati iste objekte iz razlicitih pozicija. Neke ivice I poligoni tih objekata ce se iz jednog ugla posmatranja videti, dok iz drugog nece I obrnuto.

*Bojenje vs Sencenje: Bojenje je obavezno, sencenje se moze izostaviti. Bojenje svaki px celog objekta oboji uvek istom bojom. Tek prilikom sencenja uracunavamo I svetlo sa njegovim karakteristikama.

***Kako racunar da zna sta da boji**? Kada mu stignu verteksi, on ne zna koju stranu poligona da boji (tj sta je lice a sta nalicje). Tu uskace **NORMALA**. Ako normala poligona ide ka kameri, on se vidi! Ako ne, okrenut je "ledjima" tj onda se ili ne prikzuje ili se boji nekom bojom koja je rezervisana za takav prikaz. Odbacivanje trouglova koji se ne prikazuju : **Backface Culling**. (zbog utroska resursa)

*Fragment : Skup piksela iste boje. 1 fragment ce biti predstavljen kao 1 ili vise pixsela, u zavisnosti od rezolucije.

*Rasterizacija : Prelazak sa fragmenata na piksele, moze i obrnuto (sa px na fragmente).

*Poligon:Mreza planarnih povrsina kojom se oblikuje svako telo

*Refresh rate :Obredjuje brzinu iscitavanja memorijskih lokacija (ona lokacija koja odgovara pikselu koji treba da se prikaze)

Geometrija

Zadatak geometrije je da postavi, projektuje I uredi vertekse pre rasterizacije. Geometrijske transformacije su znaci menjanje geometrije i projekcija na 2D ravan. Geometrija moze biti prepustena aplikaciji ako nema neke velike promene na sceni pa da ona izracuna tu promenu i vrati to grafickoj kartici na prikaz (npr. da napravi razliku novog proracuna i starog, samim tim procesna moc je manje opterecena ali je memorija vise zauzeta)

1. **Model & View transform**: Objekte stavljamo negde na scenu tako sto se svako teme matricnim transformacijama (neko mnozenje matrica) prevedu u svet kamere. Prevodi sva temena sa scene u neki pogled. Taj pogled je odredjen kamerom. Preuzima sve vertekse iz koordinatnog sistema scene u koordinatni sistem kamere.
   Objekti su I dalje definisani na sceni, samo mi saljemo kopije(zato sto se nista sa originalnom scenom ne menja, tj ostaje nedodirkuta i neupropascena,a s tim kopijama nesto radimo) objekata koji su "upali" u kadar na obradu. Nista se ne gubi, samo se ne prikazuje sve. Uglavnom se na Z osi nalazi kamera kako bi projekcija I clipping bili jednostavniji.
   Izlaz iz ovog cvora je Frustrum, tj deo prostora koji kamera vidi, koji sadrzi samo TEMENA.

2. **Vertex shading (shader)**: U principu "igranje" sa temenima. Moze da radi sta god hoce sa verteksima, ali NIKAKO da ih ukida ili dodaje! De sme biti destruktivan na verteksima.
   Sposoban je da trazi usrednjene normale svakog verteksa (da napravi da razlika u sencenju izmedju trouglova bude mala tj. da se lepo prelivaju senke iz kraja jednog u pocetak drugog).Moze da proracuna na osnovu definicija normala I informacija o tome gde je koje svetlo ILUMINACIJU u svakom temenu. Kod temena u istoj ravni je ista nijansa! Zato svako teme ne mora sa sobom nositi informaciju o normali, vec nosi samo informaciju o svetlu. On u principu priprema rasterizaciju, sracunavajuci boju, odbljesak, matrijal za svaki verteks. Vrsi masovne redukcije po svakom verteksu.

3. **Projekcija** : 3D Svet koji kamera vidi se projektuje na 2D ravan. Projekcija se obavlja iz **perspektive**(citav svet nastaje ili nestaje u jednoj tacki (moze 2,3 (zablja ili pticija) i vise), sto moze prouzrokovati blagu deformaciju objekata koji se prikazuju, ovako covek vidi - bolja za realizam) ili **ortogonalna** projakcija (odrzava realan odnos izmedju objekata).Izlaz je upravo taj 2D opis.

4. **Clipping** : Postoje objekti koji su delimicno upali u nas prostor. Isecamo SVE poligone na podpoligone koji upadaju u prostor kamere (prostor za prikaz). Jedini je destruktivan na verteksima! Dalje idu SAMO oni verteksi onih poligona koji su upali u projekcionu ravan, ona stara nas ne interesuju (koja upadaju u Viewport). Nije samo da se gube poligoni nego se i dodaju novi, zasto? Ako su ovi sto su iseceni otvoreni posle secenja treba ih zatvoriti pa se zato dodaju novi poligoni.

5. **Screen Mapping**: Prevodjenje verteksa (svih onih sto su upali u projekcionu ravan) u koordinate ekrana. Obezbedjuje prikaz da 2 objekta koji se nalaze jedan iza drugog na finalnom prikazu ne ispadnu "zalepljeni", vec dodaje laznu dubinu (nije obicna ravan gde su svi sabijeni, vec kao neki kuboid i on se zaprvo pojavljuje tokom projekcije). Uracunava I transparentnost. Sve koordinate su sada izrazene u procentima (od 0 do 100 ili 0 do 1 a ne od -1 do 1 sto je vezano za neku kontretnu biblioteku)! Posle ovoga je geometrija uradila svoje i sada stupa na scenu **rasterizacija**.

Primeri (Model & View transform***):***

- **Primer 1**
  Covek vozi auto. Iza zadnjeg sedista je kamera. Vidi se covek koji vozi, deo enterijera automobila i ulica kroz prednji prozor. Posto moramo paziti koji deo ulice se vidi kroz prozor, cela se projektuje na ravan! To nam daje jednu "sliku". Enterijer auta sa vozacem se predstavljati neprozirni deo druge slike, gde je prozor proziran. Sada radimo SABIRANJE ove dve slike i dobijamo konacan prikaz.
- **Primer 2**
  Prikaz kokosinjca kroz ogradu(zicu). Kokoske koje trce su na slici 1, slika 2 je zicana ograda sa prozirnim delovima, ponovo sabiranje.
- **Primer 3**
  Njihanje travcica na livadi na vetru. Spajanje mnooogo slicnih slika u stvaranju iluzije.

Dodatne napomene:

- **Object Model Space**: Koordinatni sistem u kom je objekat definisan. U njemu se obavljaju transformacije poput pomeranja temena(deformacija osnovnog objekta), resize, rotacija, itd...
- **World Space**: Koordinatni sistem scene. Svako teme svakog objekta se prevodi iz koordinatnog sistema objekta u koordinatni sistem scene(opet matricne transformacije). Po njemu se prethodni objekat pomera (nakon svih transformacija).
- **View Transform** : Prevodjenje iz koordinatnog sistema scene u koordinatni sistem kamere, tako da je kamera u koor. pocetku tog koor. sistema.
- **Eye Spacing** : ono sto korisnik vidi.
- **Frustrum** : Deo prostora koji kamera vidi. Predstavlja volumen. Zavisi da li posmatramo iz perspektive (razubljena piramida), ili ortogonalno (kvadar). Ogranicen je na prednju i zadju ivicu (minimalnu i maksimalnu blizinu koju kamera moze da vidi i sve sto je izmedju toga upalo pod uglom kamere predstavlja taj frustrum)
- Transformacije iz jednog "sveta" u drugi predstavljaju matricne transformacije. Sve transformacije se vrse pomocu matrica 4x4!
- **Fongovo sencenje**: Sencenje koristeci interpolaciju normale, pa onda primenjuje iluminacioni model.
- **Guroovo sencenje**: Sencenje koristeci interpolaciju boje, gde se boje sracunavaju u temenima poligona.
- **Ravno sencenje**: Svetlo poligona racuna se na osnovu samo njegove normale. Ostre ivice, ruzno na oko.

*Ako bi svako TEME nosilo informaciju o sopstvenoj osvetljenosti velike bi bile razlike u nijansama/svetlu, sto nije realno tako. Problem kod (lepog) prelivanja boja. Zato vrsimo usrednjavanje normala : vektorski zbir daje pravac, a intenzitet je jednak (a+b)/2. Jedan verteks moze minimalno biti deljen izmedju maksimalno 3 susedne povrsinice neke mash mreze (jer tek 3 daju **volumen**!).

*Projekcija I clipping se mogu **paralelno** izvrsavati, I u zavisnosti od zadatka moze I vertex shading.

*Pitanje : Da li ce projekcija (kao korak/proces) nestati?
Odgovor : U slicaju 3D prikaznih uredjaja HOCE!

*Pitanje : Da li ce clipping (kao korak/proces) nestati?
Odgovor : NECE!

*Pitanje : Ko ima vise posla, geometrija ili rasterizacija? Geometrija, ako objekat za prikaz ima puuuno verteksa. Manje je zahtevna kod prikaza glatkih povrsina (plastika, metal…). Problem pravi prikaz poput zalaska sunca (mnostvo nijansi, glatki prelazi, gotovo ni jedan verteks).

Rasterizacija

Trazi prostor izmedju vertexa, taj prostor deli na podprostore koji imaju istu karakteristiku boje, stvarajuci fragmente koji ce postati pikseli (1 ili vise).
Dobija trouglove definisane svojim temenima u 2D prostoru.

1. **Triangle setup**: Postavlja trouglove. Odbacivanje onoga sto nam je okrenuto ledjima I zbog preklapanja moze se praviti vise trouglova nego sto je bilo ranije.
2. **Triangle transversal**: Ponalazi fragmente u trouglovima (minimalno 1px, obicno vise).
3. **Pixel Shader**: U prethodnim koracima sracunata boja, normala, osvetljenje za svaki fragment? (mislim da je vertex, a da se ovde boje fragmenti) . Obavlja se bojenje, Ravno, Guroovo ili Fongovo sencenje, na kraju se lepi tekstura. Na ovom koraku se prikazuje sadrzaj video bafera.
   Ukoliko se prilikom bojenja javi spekularna refleksija to znaci da nam je objekat obasjan veoma jakim tackastim izvorom svetlosti da se na nekim delovima gubi informacija o boji.
4. **Merging:** Efekti sa kolima, kokosinjac, magla… Lepljenje nekih “jeftinih” slika preko rezultantne dobijamo dobre efekte uz manje napora da se prolazi kroz pipeline.

*Pitanje : Zasto ne mozemo na nivou geometrije da radimo teksutre? Zato sto su one definisane na novou **slike** a ne geometrije.

Generacije GPU :

1. **Do 1995.**: Sve je radio CPU. Rasterizacija je radila algoritamski. Algoritmi su postali veoma usavrseni, matematicki koprocisori su to dosta dobro obradjivali (nisu imali upravljacku funkciju, samo su radili matematiku). Potreba (trazenost i popularnost grafike), za sve slozenijim (sa vecom realnoscu) grafickim prikazima uslovila je da se od njega napravi prvi graficki koprocesor pa I prvi GPU.

2. **1996.** : Pojava prvog GPU. Prikazi realniji, kretnja brza. Znao je samo za mapiranje teksture. Transformacije teksel na piksel, teksel na piksel. Z bafer sluzio da proracuna koji je piksel najblizi projekcionoj ravni (manja Z koordinata), pa da taj piksel prikazuje - zaklonio je ostale (poredjenje piksela u jednoj ravni).

3. **1998.** : GPU koji razdvajaju geometrijsku I rasterizacijsku fazu. Procesor poseduje odvojene, specijalizovane instrukciju (koje pozivamo i kao normalne procesorske insturkcije asemblerskim jezikom) za jedne I druge poslove.

4. **2001. – 2002.**: Prva ozbiljnija generacija GPU.Pojavio se prvi shader na nivou procesora (znaci imaju osnovne instrukcije za njega). Postoji mogucnost upravljackih struktura, ne ide kao ranije sve staticki I u sekvenci, ali je

sve fiksno.Ne trazi se samo njegova usluga, vec pocinjemo da ga programiramo kombinujuci njegove osnove.

5. **2003. – 2004.**: Vertex shader postao dinamicki (znaci imao je dinamicke strukture, tj programiramo ga kako mi znamo danas), pojavio se prvi staticki pixel shader.

6. **2004. – 2006.**: Dinamicki I vertex I pixel shader, pisemo programe za obradu I na nivou verteksa I na nivou piksela. Ovakva promena na nivou arhitekture procesora je uvela promene I na fizickom nivou.
(Nividea GeForce 6800) : Posebne procesne jedinice za pixsel I raster shader. Pracenje finih promena u geometriji gde nema puno promena u boji optereti ce vertex shader, dok ce raster ostati besposlen. Slicni kao sa nebom, verteksa nema, a raster puca od muke. Problem nastaje pri hladjenju silicijuma! Jedan deo gori, a cela ploca se hladi, ne mozemo hladiti samo jedan deo.

7. **2006.** : Unified Shaderi! Imaju instrukcije I za vertex I za pixel shading. Neke usko specijalozovane instrukcije i tipovi podataka za jedan I za drugi sada ne postoje, teze ga je programirati, ali je masina mnogo brza. On u jednom (ili vise, ako to scena zahteva) prolazu moze da radi vertex shading pa da predje na pixel shading, uvek je isto opterecen! Temperaturu je lakse odrzavati, tj zagrevanje je isto i mozemo efikasnije da hladimo.

***Unify Shader**:
Predstavlja presek skupova instrukcija i tipova podataka tj. 1 jezik za vektorski I rasterski deo. Odvija se na istom delu procesora. Velika komponenta sa puno jedinica, deljena(zajednicka) memorija I deljeni resursi i taj deo ce uraditi posao u 1,2, ...n prolaza.
- Lose : Veci broj instrukcija I tipova podataka je sada potrebno poznavati I koristiti kako bi se obavio isti posao. Programeri morali biti vestiji.
- Dobro : Nemamo neopkupirane (slobodne) jedinice.

*GPU moze deo vremena biti neuposlen, zaduzi ga za nesto drugo!

*Pitanje : Kako mora biti zadata instrukcija da bi GPU znao da je izvrsi?
Odgovor : Zadata u matricnom obliku! Onda on moze raditi za visokim nivoom paralelizacije.

*GPU spada u **SIMD** (Single Instruction Multiple Data) arhitekture : Matrica je po definiciji MD, a GPU nad jednom matricom izvrsava mnostvo instrukcija.

*Pitanje : A kako bismo povezali GPU sa CPU I sistemskom memorijom tako da mogu da komuniciraju?
Odgovor: Imamo 2 opcije.
1.Povezi ih **SPOLJNOM magistralom**(fizicka, na maticnoj ploci).
2. Povezi ih **INTERNOM magistralom**: Najbrza je, ona je unutar procesora, sa njom komuniciraju Cash buffer-i, registri I ALU. Komuniciraju na “nivou hemije” tj napravljeni su od istog materijala (na istoj silicijumskoj ploci, iste su prirode) pa je komunikacija najbrza moguca.

*Pitanje : NVIDEA ili AMD I zasto?
Odgovor :
- NVIDEA : Bolja je saradnja! Moze da komunicira sa bilo kojim CUDA kompatibilnim centralnim procesorom! GPU ima posebnu vezu (dodatne cipove) da moze da se dozvoli zahtev od CPU da se iznajmi paralelizam i onda GPU daje instrukcije, obradjuje podatke i daje rezultat te obrade (upisuje u memoriju). Moraju postojati dva pina, jednim se salje zahtev GPU-u, koji odgovara i CPU primi odgovor (valjda drugim pinom?) i angazuje GPU.
- ATI-AMD: Brzina komunikacije je veca jer sve ide po brzoj magistrali tj, na jednoj silicijumskoj plocici su napravljeni i CPU, GPU i komunikacioni procesor). CPU i GPU rade u paraleli i GPU moze pomoci CPU-u na novou **silicijumske ploce** i to je daleko brze. Odrzavanje je bolje, jer je sve napravljeno od strane istog proizvodjaca! Ali moze da komunicira samo sa onim CPU sa kojim je ugradjen. Svaki sistem je jak onolko kolko i njegova najslabija komponenta! Tako da ako je 2 u 1 kao ovde, ako jedna radi slabije ceo sistem je slabiji.

Programske Biblioteke

1. **OpenGL** : Sire rasprostanjena, najstarija, podrzana za sve platforme…
2. **DirectX**: Microsoftov, bio deo OpenGL-a. On je skup multimedijalnih softverskih biblioteka koje podrzavaju rad cak I sa audiom, videom… Koristi se samo za Microsoft platforme.

*Pitanje : Sta je bolje, OpenGL ili DirectX?
Odgovor:
- **OpenGL** : Jednostavniji za koriscenje. Lici na pozive matematickih biblioteka sto je programerima intuitivnije. Podrzan je od velikog broja programskih jezika. Besplatan, otvoren kod. Lose je sto smo ograniceni na ono sto nam glut dozvoljava.
- **DirectX** : Brze probija nesto novo, progura tu novinu pa tek onda OpenGL to ukljuci. Lose je sto je dosta teze isprogramirati, ali nismo ograniceni.

OpenGL :

*Podrska za embeded sisteme : OpenGL ES (Ocerupan, napravile ga firme za svoje potrebe, spustili ga dole na nivo nize, da se prilagodi automobilima, ves masinama…)
*Podrska za Web : WebGl (Ocerupan, da bi mogao da se izvrsava unutar browser-a. Ako nema nigde grafickog procesora moze se izvrsavati I u CPU). Smatra se bezbednosnim rizikom jer visok nivo browsera koristi nizi nivo GPU-a.

*Mislim da se kasnije prosirio i na audio i na obradu slike kao Microsoft

***OpenGL pipeline**(slika) :
Aplikacija je spremila (posle korisnikove aktivnosti na ulaznim uredjajima) sve vertekse (moze i pixel data da dodje) I njihove karakteristike kako bi se generisao novi graficki prikaz. Napunimo Display listu sa Vetrex data (Moguce pitanje : Koji je to rezim rada? => RETAINED, jer display listu punimo sa onim sto ce biti prikazano). To ide na rasterizaciju sa kojom ce se lepiti i teksture i puni se bafer koji to treba da prikaze.

*Pitanje: Sta novo uvodi OPENGL? Ako nesto u DisplayListu stize sto nije triangle zasnovano (npr. krive zakrpe), to ide u Evaluaciju gde se te krive zakrpe pretvaraju u skup trouglova.Tj. dozvoljava da mu nesto dolazi opisano skupinom piksela(slikom), kao vertex data(skupina trouglova) ili krive zakrpe.

Ima slucaj da se vrati buffer i da se sabere sa prethodnom kombinacijom (npr. blurovanje) Nisam siguran za ovo!!!

Glut ->**Graphics Library Utility Toolkit**. Eksplozijom upotrebe OPENGL-a ljudi su omogucili toolkit koje je jos vise podogao apstrakcionu moc programiranja, tako da je lakse programirati i graficki deo (tice se iscrtavanja) i sistemski deo(tice se usluga poput otvaranja prozora za iscrtavanje, aktivaciju drajvera itd…). Za svaki od operativnih sistema ima svoju verziju tog programiranja sistemskog dela koje ce da uradi za nas. Znaci dobro je sto deo posla ne moramo da uradimo, ali smo ograniceni (u kavezu) s onim sto on omogucava.

*glutSwapBuffers() -> zamenimo front i back buffer, tj. ono sto je bilo back postaje front, a sprema se novi back. I tako u krug.

***DirectX pipeline**(slika) :

DirectX moze da radi bez ulaznog uredjaja, tj da ga simulira. Kod DirectX moramo da programiramo i sistemski deo, samim tim imamo i vise posla, ali nismo u "kavezu" kao kod GLUT-a.
Kod Vertex Shadera se pripremi teksutra da bi dalje bilo jednostavnije. Pojavljuje seGeometry shader : radi nad verteksima koji definisu 1 poligon. Output Merger : sastavi (merguje) sliku.
Ide se ka tome da mozemo sami u delove organizovati kako I cime punimo buffer-e, tj. sve mi mozemo da isprogramiramo ak oi kako zelimo ali na GPU.

*Kasnije je napravljen DirectX sa C# koji je "umotan" kao GLUT, ali ne tako efikasan. Ima vise slobode u upravljanju, ali manje fizickog rada nego klasican DirectX.

Modeli, Prostori I Transformacije

Prikaz algoritamskog resavanja, bez ulazenja u matematiku.

- U model-space mi napravimo objekte. Svaki objekat se nalazi u svom model-space.
- Objekte postavljamo negde na scenu(world-space) tako sto se svako teme mnozenjem matrice prevede iz sveta modela u svet scene. Moze doci do skaliranja, rotacije, translacije…
- Transformacije se obavljaju pomocu transformacionih matrica I one ne obavljaju samo skaliranje, rotaciju I translaciju vec I neke graficke transformacije poput projekcije, image-mappinga.
- Dalje se transformisu objekti sa scene u **pogled** (koordinatni sistem) kamere (View Transform). Napravili smo 'kopiju' (jer su originali i dalje na sceni nepromenjeni) objekata sa scene koji idu dalje na obradu. Ovo takodje radimo sa 4x4 matricama i primenjuje se na vertexe.
- Posle toga se projektuje na 2D ravan (opet matrice 4x4 se koriste), tj. na kuboid i to se normalizuje(opet matrice 4x4 se koriste) i clippuje. Clipping ne radi matricno vec algoritamski!
- Posto radimo u 3 dimenzije potrebna nam je matrica 3X3, medjutim malo smo zakomplikovali matricu kako bismo sve operacije sveli samo na mnozenje matrica(bez sabiranja npr), tj da bismo imali brz hardver (koji nije preterano inteligentan) pa samo uveli **jednu HOMOGENU koordinatu**. Prema tome, koristimo matrice 4X4 kako bismo preslikavali koordinate I slicno, vrsili projekciju itd…

Kako napraviti 3D modele:

1. Modelujucu samo njihovu spoljasnjost, a ne unutresnjost (Boundary, “Ljustura”,tj. samo granica izmedju tog dela i okoline). To su onda mash-modeli, trouglovi, cetvorouglovi…
2. Modelovanje unutrasnjosti, a spoljasnjost se sama generise. Modelovanje **punim telom** (solid modeling). Uzimamo puna tela pa ih spajamo (kao vajar) ili odbacivanjem (kao klesar). Odbacivanjm odbacuje delovi materijala koji nam ne trebaju tako da ostane samo ono sto treba (sto nam je model). Modelujemo volumen ali sa obrnute strane, krenemo od celog sveta da bi dosli do modela. Odbacivanje je blize nama racunarcima, jer delimo prostor na deo koji sadrzi model I koji ga ne sadrzi,
3. Spacijalna enumeracija : Prostor delimo na podprostore, koji delimo na podprostore itd. dok ne dobijemo samo ono sto je u potpunosti unutra ili u potpunosti napolju (homogen).

Modelovanje spoljasnjosti:

- Polygon-mash: pomocu trouglova uglavnom, jer svaki element mora biti planaran. Polygon meshjeste skup temena/verteksa takvih da se dele izmedju najmanje 3 poligona kako bi bilo moguce da se izgradi volumen. Orijentacija normale odredjuje sta je unutra a sta napolju u tom polygon mash-u. Orijentacija je odredjena redoslednom navodjenja temena. Sto vise trouglova(planarnih poligona) to je model lepsi, ali je i slozeniji matematicki pa se trose resursi. Modeleri ga nekad ne vole zato sto je glupav, tj. zele da modeluju cetvorougao, a oni moraju da to urade kao 2 trougla. Bilo bi lakse da imaju gotov cetvorougao.
- Parametric patch: parametrizovane zakrpe (pomeramo 'koncice' u povrsini i pravimo to sto zelimo). OPENGL ga na kraju pretvori u mrezu trouglova. Sluze za modelovanje 'finih' stvari.
- Implicit surfaces: Suva matematika ($\boxed{r^2 = x^2 + y^2 + z^2}$). Popularizovane od strane avio industrije. Sve sto nije matematika pravi problem. Takodje, moguce je napraviti telo simulirajuci fizicke pojave(primer : Metaballs). Cesta pojava kod modelovanja naprezanja I opustanja misica u filmovima (primer : dinosaurusi u Jurassic Park).
  Implicitne povrsine koristi programsko okruzenje Organica

*__Mash Model__: Mreza planarnih poligona opisanih verteksima.

*Pitanje : Kakav uslov treba da zadovolji verteks da bi bio dobar za poligon mash?
Odgovor : Isti kao I odgovor na pitanje izmadju koliko poligona bi najmanje vetreks trebao biti deljen?
3! Kako bi sve moglo lepo biti zatvoreno u ljusturu, volumen, a sa 2 bi bilo otooreno.

*Neplanaran poligon predstavlja parametrizovanu zakrpu!

***LOD** : Level Of Details, veci je sto je slozeniji model, tj imamo vise verteksa. Moze da se menja, zavisi od onoga sto modelujemo I treba biti prilagodjen scenariju upotrebe tog prikaza.
***Ivica** : Povezuje 2 vertexa. Dele je iskljucivo 2 poligona.
***Trougao** : Moze da predstavlja bilo koji poligon, bio on konveksan ili konkavan. Lako mu se nalazi normala (ona odredjuje koje je 'lice' trougla).
*Bitno :
- Sto su poligoni VISEG stepena, geometrija je JEDNOSTAVNIJA!
- Sto su poligoni MANJEG stepena, geometrija je SLOZENIJA! Jer manji stepen daje vise trouglova.

*Pitanje : Mi pravimo biblioteku, kako da definisemo/memorisemo normalu?
Odgovor: Uvek moramo definisati verteks(e). Uvedemo konvenciju da je redosled bitan (levi ili desni triedar) I da on odredjuje normalu. Samim tim dobijamo nesto sto je mnogo efikasnije od vektorskog proizvoda koji bi se inace koristio za racunanje normale. (Kod OPENGL je pravilo desne ruke,a DirectX je leva ruka)

*Pitanje : Koliko nam je potrebno trouglova za prikaz?
Odgovor : Onolko kolko je potrebno da opisemo sve potrebne detalje. Necemo preterivati, razmisli da li modelujes nesto u blizini ili udaljeni objekat I da li se taj objekat krece. Neces sigurno udaljenog konja koji galopira modelovati sa 2000 poligona.

Kako napraviti 3D model objekta pomocu poligona?

***Generisati skup tacaka** :

1. Rucno: Dosta zahtevno i previse rada.
2. Matematikom: Tela dobijamo tako sto rotiramo neke krive oko neke ose. Takodje mozemo izvlaciti napolje ili iznutra novi element po nekoj krivoj bez gubljenja volumena (ta povrsina se stalno dopunjuje, nikad se nece prekinuti). Za ovaj pristup traba ima osecaja kako ce nesto sto je 2D izgledati u 3D.
3. Skeniranje: lasersko ili opticko. Mozemo skenirati npr. 3D skenerom ili nasim mobilnim telefonima. Ako je objekat dobro osvetljen dovoljno je fotografisati iz vise razlicitih uglova.

***Delaunay-ova triangluacija**: Pravljenje trouglova tako sto se povezu bilo koje 3 tacke i napravi se trougao, pa se opise kruznica oko trougla. Ako u kruznicu ne upada ni jedno drugo teme ove 3 tacke mogu praviti trougao. Radi se u situacijama kada dobijemo izgenerisan skup tacaka i moramo povrsinu neku da napravimo. Problem je sto se moze napraviti takav skup tacaka da ovo ne upali pa onda mora korisnik da intervenise, tj. da ukloni ili doda neke tacke.

***Non-Uniform Rational B-splines krivama/povrsinama**: Povrsine koje se mogu kriviti

*Pitanje: Ako imamo model u OPENGL-u sa vertexima kako modifikovati da radi u DirectX?
Odgovor: Promenimo redosled navodjenja.

*Pitanje: Zasto procesor radi sa trouglovima?
Odgovor: Zato sto je planaran, moze se bilo koji drugi poligon predstaviti pomocu trouglova i normala se lako odredjuje matematicki kod trouglova (iako to nas ne zanima jer mi navodimo vertexe u redosledu i tako dobijamo normalu).

Opticko vs Lasersko skeniranje

1. **Opticko** : Npr. 3D face capture. U prostoriji imamo vise kamera koje snimaju objekat iz mnogo uglova. Zahteva da objekat jaaako dobro osvetljen! Te slike se sastavljaju I tako dobijamo teksturu objekta koju mozemo prevuci preko bilo kog modela.

2. **Lasersko** : Lasersko radi u potpunom mraku cak je i bolje da bude u mraku da sto manje svetlosti (npr. crvene) bude tu koja nije ta crvena lasterska. Funkcionise tako sto izvor laserskog zraka mora biti upravljan na osnovu

geometrije tela koje se skenira, baca liniju tackica, a kamera ce gledati tackice i na osnovu odstupanja od prave linije ce tumaciti sta je unutra, sta je izboceno i odnose izmedju susednih tacaka. Moze i da se okrece objekat pa da sinhronizujemo okretanje sa prikupljanjem laserskih tacaka pa ce se skenirati objekat **sa svih strana.**Kod laserskog mora se imati dobar osecaj za geometriju objekta i kako usmeriti ugao.
*Problem: Treba jako dobro uskladiti ugao kod skeniranja i geometriju samog objekta. Takodje ne dobijamo informaciju o boji! Npr. ako imamo crvenu boju na modelu koji skeniramo i crvena je boja laserske svetlosti on nece znati sta cita.

*Pitanje: Sta ako zelimo da napravimo neki model, kako da pocnemo?
*Odgovor: Probamo da kupimo, ako je preskupo, da smanjimo cenu, ako nista od toga ne ide onda pravimo sami.

Kako definisati trouglove pomocu vertexa:

1. Listom indeksa: Takva liste gde je svako teme opis 3 koordinate (na nivou podataka referenca). Problem je sto se neko teme moze ponoviti bar 3 puta (jer je svako teme deljeno izmedju bar 3 poligona), tj. redundantno zapisivanje.

2. Trake (strips): Korisnimo prethodno definisana temena tj. nadovezujemo se na ivicu. Traka je topoloski element. Problem? Svaki drugi poligon nece biti dobro orjentisan, kako to resiti? Ili da obrnemo orjentaciju svakog drugog tj. okrenemo mu normalu ili obrnemo navodjenje tj. redosled navodjenja ili citanja kod svakog drugog (npr. da ne budu tacke 2 pa 3, nego 3 pa 2)

3. Ventilator: Prvo memorisemo centar, a onda ostale tacke. Jako stedi memoriju, jer trouglovi dele jedno teme, tj. to koje prvo navedemo. Problem? Moramo voditi racuna o zatvaranju ventilatora. (0,1,4,6,9,3,2) -> otvoren ventilator, (0,1,4,6,9,3,2,1) -> zatvoren ventilator.

Implicitne povrsine

Metaballs: Dobre za simulaciju bilo koje tekucine i odlicne za simulaciju misica. Popularizovalo se kod filma Jurassic Park za simulaciju vretenastih misica, gde su se u jednu opnu stavile 3 loptice i one su se sirile i skupljale da bi simulirali skupljanje i sirenje misica. Preko toga se prevukla koza dinosaurusa i lepo se video taj efektat uz skromno trosenje resursa.

Kako ih napraviti? Zamisliti kao da imamo tackasti izvor zracenja nekih cestica koje se iz tog tackastog izvora uniformno i radijalno sire (znaci na sve strane podjednako ide, nema usmerenosti i istom jacinom) i onda se za neki deo prostora gleda da li u odnosu svih metaballsa da bude manji od praga (neke vrednosti) i ako je to ispunjeno to ocvrsne. Npr imamo loptu koja je 'sama' i iz njenog centra idu cestice dok budu manje od praga. Sto vise povecavamo prag to je lopta veca, sto vise smanjujemo to ce lopta biti manja, jer ce brze dovoljan broj cestica doci do praga. Ako dovedemo dve loptice jednu do duge onda ce jedna imati uticaj na drugu. Tako da ce najveci doprinos jedne loptice ka drugoj biti u jednom delu i brnuto i sto ih vise priblizavamo one ce se dodirnuti u jednoj tacki i ispada kao da se nesto deli, razdvaja. Tako da sa promenom precnika i praga mi simuliramo razlicite efekte I uticaje (npr. disanje).

Parametrizovane zakrpe

***Prava I Kriva:** Ako poznajemo 2 tacke neke prave, moze se pomocu nagiba odrediti jednacina prave. Ako nam treba jednacina krive, moguceje dobiti je nekim transformacijama (najcesce Furijeove trans.). Sta cemo ako imamo veoma komplikovanu saru koja imam geometrijski sablon koji se ponavlja (npr. sara na tepihu)? To je jako tesko opisati matematicki, a jos teze da racunar to treba da iscrtava u svakoj iteraciji.
Resenje daju param. zakrpe, jer sada ne zapisujemo sve pomocu jedne jednacine ponasanja zavisnosti jedne koordinate od druge (y = f(x)).

***Katodna cev:**Zadatak je da bude na jako velikom negativnom naponu (elektroni se gomilaju na katodi i stvaraju oblak). Sa druge strane imamo anodu koja stvara pozitivan napon (iste jacine) tj. manjak elektrona. Negativni elektroni ce

krenuti da 'trce' do anode i dobijaju kineticku energiju i bukvalno ce probiti anodu i udariti u centar zaklona na kom je namazan fosfor (lako menja energetske nivoe). On oslobadja tu fotone (zelene frekvencije) energije sto je dobio da bi presao na visi nivo onolko dugo kolko mu treba da se vrati u normalu (normalan nivo). Ako zelimo da konstantno svetli mi ga 'udarimo' da zasvetli i pre nego sto gledalac moze da primeti gasenje, mi opet 'udarimo' (osvezavanje). Sta ako je neka anoda npr. gornja na vecem pozitivnom naponu? Ona ce ici gore, pa mozemo da se igramo i dajemo veci napon nekoj drugoj(po moze ici i dole i levo i desno, a ako damo gore i desno podjednako visi pozitivan napon od druge dve onda ce ici u dijagonalni cosak). Samim tim mi crtamo krive tako sto menjamo napon na skretnim plocama. Leve i desne anode su x osa, a gornja i donja y osa. Iscrtavamo tako sto imamo jednu funkciju po x i drugu po y i one naravno moraju biti **sinhronizovane (po vremenu t)**. Katodna cev i dalje daje najbrzi prikaz, ali po svim drugim karakteristikama je prevazidjena.

Parametrijske zakrpe: Zasto zakrpe? Zato sto nesto sto je npr. u 2D ne prikazujemo kao jednu jednacinu prve dimenzije od druge nego pravimo 2 jednacine. Sada se za opis jedne linije (u 2D prostoru) koriste 2 jednacije treceg reda, kako bi se preciznije definisala brzina promene! Za 3D koristimo 3 jednacine. Sto je veci stepen veca je preciznost, detaljnost, tj. iskrivljenost (na arhitekturi se koriste 4. ili 5. red). x(u,v), y(u,v), z(u,v) -> svaku definisemo sa dve nezavisne promenjive, tj. zajednicko usinhronizovano kao 'tkanje' dveju grupa krivih po 3 promenjive. Na presecima njih su **kontrolne tacke** i njih mozemo vuci i gurati i samim tim kriviti povrsinu (koja je elasticna). Splajnovi(krive prikazujemo po segmentima, isprekidano)-> Sto su cvrsce povezane svaki najmanji pokret jedne krive prenosi se na drugu (kao da je sve jedna kriva).

Spacijalna Enumeracija

Deljenje prostora:za 2D su QUADTREE-s (svaki roditelj ima 4 dece, znaci idealno za **ravan**, memorisemo samo ono sto nas interesuje), a za 3D su OCTREE-s zato sto sada radimo sa **prostorom**(4 gore i 4 dole, listovi opisuju model u elementima prostora).

Modelovanje punim telom

Construct solid geometry:prvi nastao, zbog potrebe za softverom koji ce pomoci u modelovanju sto boljeg motora sa unutrasnjim sagorevanjem za General Motors, za promene oblika klipa I druga istrazivanja. Ostvaruje se pomocu:

1. Sprovodjenjem skupovnih operacija nad osnovnim geometrijskim telima (unija, presek, razlika…)
2. Sweep : Uzmemo telo i definisemo trajektoriju kretanja kroz prostor i svaki polozaj koji zauzme to telo ce 'ocvrsnuti'. Tako nastaje novo telo koje moze biti veoma slozene geometrije (npr. da bismo napravili konus rotiramo trougao).
3. Solid Extrusion : Neku osnovu izvacimo koja po osnovi izvlacenja zauzima nove polozaje I kreira nova tela.
4. Cut/Slot Extrusion : Izvlacenje rupe
5. Revolved Extrusion
6. Revolved cut : Modelovanje unutrasnjih cevi
7. Loft : Modelovanje novih modela od postojecih pomocu loft linija. Tj. spojimo 2 tela loft linijom, a racunar napravi model koji ima 2 osnove razlicite.
8. Shell : Eroditanje tela dok mu ne ostane samo ljustura.

3D transformacije

Na sceni su prisutni modeli, kamera, svetlo… Nekim matricama smo postavili objekte na scenu (slika, matrice M_n). Matrizom V smo postavili kameru na scenu. Kada smo kameru postavili negde, podraumeva se da cemo kroz nju I gledati, usladjuje ViewTransform gde sve prevodi u koordinatni sistem kamere, sto olaksava sve transformacije koje usledjuju potom.

3D transformacije: ModelView transformacija V^-1 * M koja stavlja sve u koordinate kamere. Zasto -1? Da bi sve bilo definisano u odnosu na novi koordinatni pocetak. Pored translacije, rotacije, itd. mora se spomenuti i projekticja koja nam daje kuboid koji na frontovima daje sliku projektovanih temena sa laznom dubinom (da vidimo ko je ispred, ko iza,

ko je transparentan, pa to kasnije lakse mozemo rasterizovati). Takodje imamo i slanje u image space tj. deo prostora spremnog za rasterizaciju. Svi oni koriste matrice istog oblika4x4.

*Translacija - sabiranje 2 vektora sa dimenzijom kolika je I dimenzionalnost prosrota.

*Smicanje - povucemo na neku stranu.

*Mirroring - objekat spojima sa xy povrsinom.

*Skaliranje – Mnozenje matrice I vektora. Ne samo da ce objekat povecati/smanjiti nego ce pobeci/pribliziti se tacki u kojoj se nalazi objekat. Zato skaliranje vuce translaciju jer moramo taj objekat pomeriti tamo gde je je originalno bio.

Homogenizacija:
Za 2D prostor: Multipliciranje 2D ravni po nekoj osi I svaka slika tog 2D prostora je od prethodne udaljenja za **w**, tj izmedju njih postoji ekvivalencija. Prema tome, n-ta slika tacke (x, y, 1) ce biti (nx, ny, n). Moze se primetiti da kako se w smanjuje ka 0, ispadne da sve nestaju u jednoj tacki I moze se smatrati normalom.
Pretvaranje obicnog 2D prostora (ili 3D prostora) u homogeni sa dodatnom koordinatom. Dodali smo novu koordinatu (veci utrosak memorije) ali zato se sada sve transformacije vrse kao matica*vektor (lako je za paralelizovanje). Dobra stvar je takodje sto na osnovu izgleda matice mi mozemo da zakljucimo o kojoj transformaciji je rec. Normalizaciju vrsi onaj element matrice u cosku. Sto vise sve tezi ka 0 sve nestaje u jednoj tacki tako da je to kao normala u homogenizovanom prostoru.

*Pitanje1: Da li rasterizacija zahteva matrice? Odgovor : Ne! Ona samo trazi piksele i puni frejm buffer.

*Pitanje2: Kako tacke jednog pravougaonika prevesti u drugi da bude bez bezanja i priblizavanja? Odgovor: Skalirati na zeljenu veliciju i zatim pomeriti (translirati) tamo gde treba da se nalazi.

***Pipeline transformacija:**

- Transformed eye coordinates: koordinate u svetu kamere.
- Clip coordinates: u projektovanom svetu (2D ravni) i usput klipovano. Ne koristi matrcce, iskljucivo algoritamski
- Normalized device coordinates: Normalizovan kubiod, da bude spreman za rasterizaciju.
- Viewport transformation: Sprovodjenje normalizovanog kuboida da odnos visine i sirine bude skaliran na odnos visine i sirine prozora koji ce da prihvati piksele koji ce se pojaviti u fazi rasterizacije.

Pitanje: Ko je doprineo da idemo mnozenjem matrica u transformacijama? Odgovor : Homogenizacija koordinata.

Pristupi projekcije:

***Raytracing algoritam:**
Napredna grafika,tj. velika realnost u prikazu, nema klasicnog pipeline-a, prati prirodu, tj. sta se zaprvo desava u prirodi. Algoritamski komplikovano i samim tim velika procesna moc je potrebna.

Kako mi zaprvo gledamo? Mora postojati svetlost, koja pada na povrsinu koju gledamo, odbije se svetlostni zrak koji dospe u nase oko, koje ce biti nadrazeno i samim tim mi to vidimo.

Postupak: Svetlost pada na objekat, odbija se i ide prema kameri. Krece od kamere, tj. sta ona vidi, pa onda od nje polazi svetlost (raytracing - pracenje sraka) i on padne na objekat. Na prvi objekat koji padne mi onda dobijemo informaciju sta je on, koje je boje, itd. Posto treba da ide na 2D ravan projektovano za svaki buduci piksel na 2D ravni tolko ce se ukupno zrakova emitovati. Svaki zrak prolazi kroz odredjeni piksel, putuje i sta prvo 'pipne' njegovu boju uzrokuje (tj. pozivanje iluminacionog modela da sracuna kolka je boja tog materijala na tom intenzitetu svetlosti) i tu vrednost boje koje je dotakao vraca kao boju tog piksela.

*Problem: Da li zrak kad dotakne neku povrsinu sme da odmah uzrokuje boju? Ne, jer ta boja ne mora da bude bas 'prava' boja, jer se taj objekat moze nalaziti u tudjoj senci. Tako da prvo mora zrak da proveri da li se ta tacka na koju je pao nalazi u necijoj senci.

*Pitanje: Kako znamo da smo u necijoj senci? Odgovor: Ako postoji objekat izmedju trenutnog objekta i izvora svetlosti. Postupak: Zrak ide iz te tacke na koju je pao ide prema svim izvorima svetlosti, tj. putuje nekom linijom do tackastog izvora, i proverava da li na toj liniji ima nekog objekta. Ako da onda proracunava senku (tj. dodaje potreban nivo sivog na boju).

*Problem1: Ukoliko postoji refleksija nekog drugog objekta. Resenje: Ne idemo samo prema svakom izvoru svetlosti, nego idemo i prema svakom ogledalu, svakoj uglacanoj povrsini koja moze da postane dodatni izvor svetlosti (ne pravi preveliku senku, ali je dovoljno uglacana da se mora uzeti u obzir i sracunati).

*Problem2: Objekat na koji naletimo moze biti transparentan (svetlost prolazi kroz njega) , pa onda mi ne racunamo samo njegovu boju nego moramo nastaviti iza njega i trazimo objekat koji nije transparentan i onda gledamo uticaj senke onog koji je transparentan i onog iza njega koji nije. Moramo uzeti i u obzir cinjenicu da se iza transparentnog objekta na osnovu njegovog materijala i geometrije slika dodatno prelama.

*Problem3: Sta uraditi s zrakom koji putuje i nikog ne dodirne? Imamo neku distancu, posle koje pretpostavimo da vise ne moze da da uticaj, pa onda zanemarimo.

*Problem4: Sta ako je objekat jako daleko? Sto zrak vise putuje on se siri i samim tim moze da pogodi i vise objekata i onda treba traziti srednju vrednost za sve te objekte (zato kada vidimo objekat u daljini vidimo ga mutno).

*Pitanje1: Kako utice na pipeline? Odgovor: On ne samo da projektuje, vec i kompletno uradi **rasterizaciju** za nas, jer prolazi kroz svaki piksel(samim tim dobijamo veliku ustedu)!!!

*Pitanje2 : Ray-tracing moze da radi I ortogonalnu projekciju, kako? Tako da zrak krece iz svakog piksela normalno na ravan projektovanja (svi zraci su paralelni). Inace svejedno moze da radi prijekciju iz perspektive.

Radi i clipping!!!

COD- centar projekcije (tu se nalazi kamera). Mi projektujemo sve sto je upalo u frustrum, tj. izmedju near(prednje) i far(zanje) ravni projekcije na prednju ravan. Kada se isprojektuje mi dobijamo kuboid koji mora da se normira na vrednosti od 0-1 ili -1-1 u zavisnosti od graficke biblioteke koju koristimo. Te normirane koordinate viewport transform (translacija i skaliranje) prevede na polozaj i odnos visine i sirine prozora na kojem treba da se rasterizuje.

Ortogonalna projekcija: ide ortogonalno na ravan projekcije i uzima uzorak objekta. Tacnija je i ispravnija od projekcije iz perspektive jer ostaju odnosi neizmenjeni.

*Pitanje: Kada koristimo ortogonalnu, a kada iz perspektive? Odgovor: Kada zelimo da pokazemo kako se nesto vidi u nasim ocima onda koristimo iz perspektive, a kada zelimo da pokazemo detalje, tj. vise inzinjerski onda koristimo ortogonalnu.

***Clipping**:Moze se obavljati zajedno sa projekcijom. Radi se algoritamski. Jedini je destruktivna operacija u geometriji, jer ima pravo da odbaci, odnosno kreira neki novi vertex sa ciljem da se prikaze odsecen deo onoga sto je upalo u viewport, a ono sto se ne vidi da odsece.Moze da se desi na neki nacin da se odsece objekat na dva njegova dela koja upadaju u viewport i oni su odvojeni, ali clipping mora da odrzi povezanost, tj. da se zna da su ta dva dela zaprvo jedan objekat. Zadatak je da se sve to uradi uz sto manje racunanja. U fazi smo geometrije tako da imamo samo temena!!! Ono sto se nalazi izmedju temena ce se naci u rasterizaciji. Prvo idemo od linija pa ka kompleksnijim objektima!

***Cohen-Sutherland**: Prozor (tj. canvas za gledanje) ce uvek biti podeljen na 9 podprostora. Imamo mali prostor 'izvan' ekrana (koji se ne pokazuje korisniku) koji nam omogucava da cuvamo konektivnost. Svaki deo podprostora kodiran je tetradama (4 bita). Najvisi bit govori dal smo ispod (0) ili iznad(1) ymax. Sledeci bit govori dal smo ispod (1) ili iznad (0) ymin. Sledeci bit govori dal smo levo (0) ili desno (1) u odnosu na xmax. Poslednji bit govori dal smo levo (1) ili desno (0)

u odnosu na xmin. Svaki podprostor ima svoj jedinstven kod (tetradu). Svi ti bitovi se lako sracunaju. Najveca je usteda u vremenu provera! Za svako teme (2n temena) svih linija (n linija) mi prevedemo koordinate iz xy u tetrade (znaci iz 4n koordinata nego 2n tetrada).

Pitanje1: Kada je linija trivijalno unutra? Odgovor: Kada su tetrade oba temena 0000 (uradimo logicko | i dobijemo 0000).

Pitanje2: Kada znamo da je trivijalno napolju? Odgovor: Kada uradimo logicko & oba temena i dobijemo razlicito od 0000.

Pitanje3: Sta ako nije trivijalno ni unutra ni napolju? Odgovor: Npr imamo 0100 i 0010. Logicko & nam daje 0000, a nisu obe 0000 tako da moramo da seckamo tj. delimo liniju tako sto prolazimo kroz viewing prozor i trazimo presecne tacke i segmentiramo na deo koji je van i deo koji je unutra. Ne mozemo da koristimo presecnu tacku sa linijom jer ona nema nikakvu tetradu, zato uzima po y-u gore i dole i po x-u levo i desno i od tacke pravi dve tacke. Tako pravi segmente. Bas kod ovog primera se desilo da je nakon seckanja ustanovljeno da linija nije prolazila kroz view-ing prozor, tako da smo uzaludno delili. To je minus algoritma, sto ne mozemo znati unapred da li ce linija preseci view-ing prozor, ali se rade logicke operacije pa nisu veliki gubici.

Problem: Zbog secenja, linija moze da promeni nagib. Kako ovo resiti? To treba vizuelno 'zamazati' tako sto cemo apriksimirati boju piksela sa strane (oko linije) kao 60(70)% boje linije i 40(30)% boje pozadine. Ovo se zove aliansing (zamucuje). Obrnuto je anti-aliansing (tj. da se ovaj efekat ukloni).

***Sutherland-Hodgman**: Klipuje poligone (zatvoreni polyline, tj. grupa linija gde je prva tacka prve linije istovremeno i poslednja tacka poslednje linije -> zatvoren poligon) bez obzira na njihov oblik. Radi sa svim poligonima (konveksan, konkavan), tako sto ih sece po linijama kliping prozora, tj. seku se linije koje formiraju taj poligon. Uvek vodimo racuna o tome sta nam je unutra, a sta spolja u odnosu na tu liniju. Ako idemo obrnuto smeru kazaljke na satu sve sto je sa leve strane je unutra i to pamtimo, a sve sto je sa desne strane te linije zaboravljamo. A ako linija sece osu po kojoj idemo, nadjemo presecnu tacku i memorisemo je. Ici po liniji i seci znaci da idemo od startnog temena tog poligona i trazimo za svako teme poligona po osi kojoj secemo dal je napolje ili unutra. Znaci onda memorisemo samo temena od interesa.Kad obidjemo citav poligon po jednoj osi onda to sto smo isekli i zadrzali gledamo po drugim osama (ono sto smo isekli po jednoj osi predstavlja ulaz za secenje po drugoj osi).

***Konkavan poligon** : Jeste onaj gde postoje neke (bilo koje) 2 tacke izmedju kojih kada se povuce linija postoje neke tacke te linije koje ne pripadaju unutrasnjosti poligona.

Culling: Odbacivanje (zanemarivanje) po nekom kriterijumu (ako nisi upao u frustrum) . View frustrum cullling ->Ono sto frustrum culling izabaci napolje, to nece ici u clipping, jer se to desava na novou citavog poligona (culling nije destruktivan tj. samo kaze dal je poligon ceo napolju ili unutra). Desava se na nivou geometrije. Clipping onda dalje radi sa poligonima koji su unutar frustruma (ili delimicno ili skroz), tako da culling skracuje posao clippingu. Back-face culling (ne prikazuje trouglove obrnute rotacije) desava se na novou rasterizacije (triangle setup). Occulsion culling se moze raditi i u fazi geometrije i u fazi rasterizacije u zavisnosti od primenjenog algoritma.

Skrivanje: Ili skrivanje nevidljivih ili prikazivanje vidljivih povrsina. Zasto se radi? Ako ne skrivamo povrsine korisnik nece znati u sta gleda! Pre toga se projektuju sve sve tacke i ako ne skrivamo one koje su iza nekih korisnik ce se zbuniti i nece znati sta da gleda. Takodje da bismo ustedeli resurse. Spada u jedan od vecih poslova.

***3D Clipping** : Slicno kao I u 2D, samo sada imamo 27 podprostora I svaki podprostor se sifruje sa 6 po bita (javljaju se prednja I zadnja ravan).

Skrivanje povrsina

*Pitanje : Zasto?
Ako ne skrivamo nevidljive povrsine korisnik nece znati u sta gleda (npr noga od stola ili lampa iza fotelje bi iskocili napred)! Isprojektovace se sve sve presecne tacke, necemo moci proceniti u sta gledamo, pa jeje realnost narusena. Razlog su I performanse, tj. sto da iscrtavamo nesto sto se ionako nece videti.

***Vrste culling-a :**

1. **View-frustrum culling** : Skrivanje svega sto nije upalo u view-frustrum kamere. Desava se u geometriji.
2. **Occlusion culling** : Ne prikazivanje objekata koji su zaklonjeni drugim objektima. Moze I u geometriji I u rasterizaciji, zavisi algoritma.
3. **Backface culling** : Skrivanje delova koji pripadaju pozadini objekta. Desava se u rasterizaciji, u triangle setup-u.
4. **Portal culling**: Podrazumeva da imamo neki portal (vrata) unutar nekog podrpostora kojem je jedina veza sa spoljnim prostorom upravo taj portal. Samim tim ako objekat nije vidljiv kroz taj portal on se sakriva.
5. **Detail culling**: Ako je nesto premalo da iz ugla kamere onda se ne prikazuje, a ako bi mu kamera prisla onda bi se video.Moze I u geometriji I u rasterizaciji, zavisi algoritma.

***2 pristupa za uklanjanje nevidljivih povrsina:**

- Object space : Sprovode sepre rasterizacije zato sto je na novou objekata (geometrije).
- Image space : Sprovode setokom rasterizacije zato sto se desava na novou slike, koja je 2D skup piksela, a pikseli se pojavljuju tokom rasterizacije.

Object-space tehnike

***Painter's algorithm**: Kako slikar slika? Nacrtace prvo najdalje objekte (nebo, planine), pa blize (kuca, pas). Takoreci igramo se sa slojevima (layerima), tj pomeramo, ih kopiramo, itd. Redosledom layera definisemo ko ce koga preklopiti. Object space tehnika je, crta citav objekat od jednom.
*Problem? Ispreplетenost! Na primer imamo par koji se grli, svaka osoba je poseban objekat. U takvim situacijama jedan objekat je malo u jednom layeru, a malo u drugom i isto to vazi i za drugi objekat. To se resava tako sto se objekat deli, ali onda on nije citav objekat vec jedan objekat sastavljen iz delova. Problem kod animacije npr. ako zelimo da ga citavog animiranog, a on je izdeljen.

***BSP trees**: Binary Space Partitioning tree. Cesto u kombinaciji sa Painter's algorithm-om. Particionise 2D prostor na dva dela. Radi tako sto deli prostor u odnosu na neki objekat na napred i nazad. Prikazuje prvo ono sto je iza objekata, a onda ono sto je ispred. Ako se pri tom deljenju nadje neko ko je jednim delom ispred, a drugom delom iza onda se taj objekat presece na 2 dela, tj. na deo objekta koji ide ispred tog objekta i na deo objekta koji ide iza tog objekta. Ako se napravi novi objekat na sceni, mora se generisati novo stablo!!!

*Pitanje1: Odakle krenuti? Odgovor: Nema pravila, ali praksa kaze da je najbolje krenuti od srednjeg objekta (onog koji je u sredini scene).

*Pitanje 2: Cemu sluzi izgenerisano BSP stablo? Odgovor: Sluzi da znamo kojim redosledom da iscrtavamo! Za jedan raspored scene se pravi stablo I moze se koristiti za generisanje razlicitih pogleda.

*Pitanje 3: Od cega jos zavisi redosled prikazivanje? Odgovor: Od kamere (pogleda). Ako kamera gleda u lice, onda treba prvo iscrtati sve koji su iza, pa objekat, pa onda one koji su ispred. Ako kamera gleda u ledja, onda treba prvo iscrtati sve koji su ispred, pa objekat, pa onda one koji su iza.

*Pitanje 4: Sta ako kamera ne gleda ni napred ni nazad nego u stranu (bok)? Odgovor: Mozemo krenuti ili desnim ili levim obilaskom stabla (nije bitno).

***Portal culling**: Specijalizovan za portale (vrata, prozor, itd.). Ako na sceni imamo neki zatvoreni podprostor, koji je malo otvoren, tj. ima mali otvor (portal) i onda u unutrasnjost tog prostora mi gledamo samo kroz taj otvor. Tu opet napravimo stablo toga sta se vidi. Morale bi da se prave projekcije (da znamo sta je upalo u pogled preko portala) i posto to moze da uzima puno procesne moci, naprave se projekcione slike i one se koriste u nekom opsegu kretanja da ne bi morala u svakom malom pokretu kamere da se pravi citava projekcija (ubrzanje). Ogledalo takodje izigrava portal, ne pravi kao vrata nego inverzni.

*Pitanje: Kako uraditi projekciju od ogledala? Odgovor: Na ogledalu napravimo kameru (virtuelnu) i to sto ona vidi zalepimo na ogledalo.

*Image space tehnike skrivanja rade na nivou piksela i samim tim vec obavljaju rasterizaciju.

Image-space tehnike

***Z buffer**: Buffer koji rasterizuje i skriva (vise nije dominantna tehnika za rasterizaciju). Predstavlja niz koji u sebi cuva vrednost piksela na nekoj predefinisanoj lokaciji po jednoj ravni preseka (najjednostavnije ga je zamisliti ako se kaze da on cuva jednu liniju rasterskog prikaza, jedan scan line koji sece 3D prostor po ravni koja odgovara tom scan line-u). Jedan piksel cuva vrednost (boju) najblizeg piksela po tom scan line-u. Inicijalno svi imaju vrednost -1 (najvece, tj. najudaljeniji Z u kuboidu koji je dosao posle normalizacije i projekcije), tj. svi idu u beskonacnost (jako jako su udaljeni). Treba ici od objekta do objektaI za njega proveravati za svaki piksel njih dal je najblizi ravni, ako jeste onda je interesantan (pobedjuje onaj kojie je najblizi) , u suprotnom nije. To sto se prikazuje je zaprvo sa **faktorom umanjenja (svetlosti) boje**, zato sto je ono sto je udaljenije je zaprvo prikazano tamniji (imaju manje svetlosti u sebi, tj. boje koja dolazi do nas, jer su fotoni koji su putovali od daljeg objekta duze putovali do naseg oka i vise energije izgubili nego od blizeg objekta).
Red za proveravanje objekata na sceni odredjuje aplikativan deo (korisnik ili softver ako je automatski) po redosledu generisanja.
*Svaki graficki algoritam mora biti nezavisan od redosleda definisanja primitiva, tj. redosled primitiva moze biti proizvoljan, samim tim nekada ce dati brzi odgovor, nekada sporiji. Ako u jedan piksel upadne malo jedne, a malo druge boje onda se radi aproksimacija, tj. aliansing. Ne zna za transparentnost i to je losa strana algoritma. Dobra strana je sto je brz (lako se proverava ko ima vecu vrednost) i dobar je za paralelizovanje.

*Pitanje: Kako resiti problem transparentnosti? Odgovor: Napravimo niz nizova (polu dinamicka struktura, nazubljen niz) koji pravimo za svako preklapanje. Onda cuva preseke i vrednosti u svakom od elemenata i sortira po indeksima Z buffera. Ako je element transparentan, memorise indekse, obicno vrednosti Z buffera, pa za te vrednosti proracunava boju. Ovo je **alfa buffer**. Memory menagement je malo komplikovaniji (zbog alokacije memorije za dodatne nizove) da ne bi bilo curenja memorije, tako da rezultat paralelizama nije tako efikasan kao kod Z buffera!

*RayTracing se moze svrstati u image-space, jer radi I rasterizaciju.

***Iluminacija:** Sracunavanje svetla, tj. da se odredi boja nekog piksela (za svaki piksel je cista fizika i to radi ray-tracing).

Teorija boja

Modeli boja: Teoretsko objasnjenje kako se boje kreiraju, kako ih covek prima. Nemamo jos tolku racunarsku moc da simuliramo toliko veliku kolicinu fotona (koja dolazi do covekovog oka kada se svetlost reflektuje od nekog objekta). Svetlostni talasi su elektromagnetni (oni koji su u vidljivom delu spektra). Oko ih prima i registruje od crvene do ljubicaste, preko zute, zelene i plave. 3 receptora u oku imamo za boju! Boju treba opisati tako da i masine napravile slike i covek tu sliku video korektno i zato je nastala teorija boja koja modeluje boju na dva nacina u zavisnosti od toga dal je mediji koji nosi boju:

- Aditivan (sabiraju osnovne boje kreirajuci boju)
- Substraktivan (oduzimaju boje koje nisu bitneod bele i reflektuju samo one boje koje su bitne, koje mi gledamo)

Bela boja je skup svih boja, a crna je odsustvo bilo kakve boje ili svetla. Ekran i beam projektor, tj. svi elektronski uredjaji rade aditivno. Suptraktivni medijum je papir, on je bele boje, svetlost pada na njega, ono sto je napisano je neke boje, reflektuje se svetlost od njega, prolazi kroz to mastilo koje oduzima neke komponente i izlazi ono sto je ostalo. Takodje je i refleksioni medijum zato sto mi reflektujemo belu svetlost od koje cemo oduzimati boju. Imamo i substrakciono-transparentne medijume kroz koje prolazi bela svetlost (grafoskop). *Dijapozitiv (isto substrakciono-transparentan) je plastican film na fotoaparatu koji se razvije u mracnoj prostoriji (da ga ne bi poremetila stvarnost) tako sto hemijom 'ucvrscuje' promene tako da one ostanu trajno zabelezene.

RGB je samo za aditivne medijume, jer sabiraju komponente boje. A za supstrakcione medijume imamo citavu familiju modela boja, jer razlicitih nosaca ima vise.

RGB: Jednostavan za matematicku obradu (to se vidi pri obradi i analizi slike, tj. prepoznavanju patterna) i za objasniti. Svaka boja je definisana u trodimenzionalnom prostoru gde ordinanta odgovara jednoj komponenti. (000 je crna, 111 je bela). Problem je sto HCI nije dobar, problem je u nacinu predstavljanja boja da covek moze lako da izabere boju. Najcesce se predstavlja kockom, gde se klizacem bira ravan iz kocke. Da ne bismo morali beleziti ravan na kojoj se nalazi, navodi se samo intenzitet (procenat) svake komponente. Iako deluje linearno po objasnjavanju, veoma je nelinearan (npr. da li da prikazujemo po ivici ili diagonali kocke? nemamo izti broj boja po svakoj!).

CMY: Cijan (plava i zelena), ljubibicasta (plava i crvena) i zuta (zelena i crvena). Propusta po dve boje od RGB i odlicno radi za supstrakcione medijume. Npr. zelimo da dobijemo lubicastu, samo pljucnemo ljubicastu, a ako hocemo plavu da dobijemo, pljucnemo ljubicastu i cijan. Plava i crvena ulaze u cijan, koji je kombinacija plave i zelene i iz cijana izlazi samo plava (tako sto se bela svetlost reflektuje se od papira i prolazi kroz to i oduzmu se komponente koje se ne vide, tj. reflektuju se plavi i crveni fotoni i ulaze u plavu i zelenu gde su zajednicki plavi fotoni). Izbija po dve klase fotona za razliku od RGB-a koji izbija jednu. Lakse je napraviti ovaj dvostruki filter, nego jednostruki (zato sto je siri, a lakse je napraviti nesto sto je sire nego nesto sto je strozije). Problem je sto kada pravimo mastilo, ne mozemo napraviti idealno mastilo tako da ce u ovom nasem primeru cijan propustiti i malo crvene boje. Kada bi smo slozili sve tri komponente trebali bismo dobiti crnu boju, ali je na zalost ne dobijamo jer ce uvek procurati treca komponenta, pa ce na kraju izaci slaba bela boja. Ovo je otkriveno kasnije, pa su do tog otkrica slike bile ne tako kvalitetne (nisu bile dobrih kontrasta). Ovaj problem se resava tako sto smo dobili **CMYK** model kod kog se doda jako malo crne boje, dovoljno da oduzme to propustanje boja. Ovo isto nisu tesko razumljivi modeli, ali opet korisnik tesko moze da izabere boju od onih koje su ponudjene.

HLS, HSV, HSB: Zbog problema proslih modela psihilozi su pitani kako covek zaprvo bira boju. Po njima covek izabere osnovnu boju, pa izabere kolko je ona saturisana (kolko je puna, zdrava, tj. kolko je ona izrazena u odnosu na druge boje, dal se vidi **samo ona**) i kolko je osvetljena sa belom svetloscu. Ako je boja manje saturisana onda je odnos nje i ostalih boja koje su prisutne mala, pa je bledja (bolesna). Ako puno bele svetlosti bude reflektovano od platna onda je jako osvetljena boja.
Predstvaljene su preko cilindra: u zavisnosti od ugla biramo boju, poteg (kolko je daleko od osnove) govori kolko je saturisana, a visina govori kolko je osvetljena. Ovaj model je pokazao da ljudi na ovaj nacin mnogo efikasnije biraju boje (tj. uspeli su da izaberu vise boja nego RGB-om i CMYk-om).

Pitanje: Zasto je cigra, a ne cilindar? Odgovor: Zato sto ako nema svetla (crno je) onda nista ne vidimo, a ako je previse belo ne vidimo ni to jer takav intenzitet beline ne vidimo.

LAB: Oko funkcionise tako sto 3 cilindra prepoznaju boje (zelene, crvene i plave). *Pri prepoznavanju plave pomazu i crveni i zeleni cilindri pa je pitanje da li postoji jos nesto sto ne mozemo da opisemo! Ti cilindri su skoncentrisani oko zute mrlje (centar vidnog polja, uvek fokusiran tamo gde gledamo), nase oko jako brzo pomera zutu mrlju i iz tackica fokusiranih boja sastavi sliku svega sto vidimo. U centru vidnog polja najbolje vidimo boje, a kako idemo ka periferiji tako se vise fokusiraju stapici, a cilindra je sve manje i manje i bolje vidimo svetlo.

Po ovom modelu ne ide svakom od ovih 3 clindra i stapicima informacija o RGB komponentama i svetlu nego idu razlike, tj. ne stize crvena, zelena i plava nego razlika kolko je to izmedju njih. Samim tim mi obradjujemo sliku na osnovu informacijama o ovim razlikama i osveljenja. *Ocni zivac nije samo zaduzen za transport neuro-signala nego vrsi i obradu. Ali kasnije je utvrdjeno da te razlike ne pravi ocni zivac nego, nego se to radi u jednom delu mozga koji vrati te razlike korteksu na obradu.
LAB model se sastoji od razlika (nema ekstremno plave i zute, niti ekstremno crvene i zelene, nego nesto sto je izmedju plave i zute, nesto sto je izmedju crvene i zelene). To su dve osnovne boje na osnovu kojih radi televizija, tj. televizijski prijemnici rade na osnovu ove dve color informacije i informacije o osvetljenosti, jer je bilo tesko shvatiti (nije bilo moci) analognim signalom RGB i svetlo (4 velicine), nego se prenosila boja preko ovih razlika i osvetljenost (3 velicine). Danas digitalnim signalom prenosimo RGB (neku njegovu malu modifikaciju) i svetlo da bi bilo prijatnije za oko.

Ovaj model je najbolji za izbor boja, jer obezbedjuje najveci izbor.

Pitanje: Kako cemo memorisati (koji model boja koristiti) slike na internetu? Odgovor: Ima nesto sto se zove web-safe paleta i svaki browser mora jednako da predstavi svaku od tih boja (ima ih oko 200). Nemoramo sve boje prenositi nego ako je rezolucija tackice dovoljno mala, susedne boje covek nece primetiti svaku zasebeno, nego ce ih izmesati i napraviti od svake od njih **mesavinu boja (jukstapozicioniranje)**. Tako da ako imamo 200ak boja mi sa mesavinom boja mozemo simulirati druge boje. Tako je usteda memorije 10ak puta veca! ->**Dithering**
Dithering ima I neke lose strane. Ne treba da se koristi kada nam je jako vazan nivo detalja na slici, npr. rendgenski snimci.

Sta ima uticaj da mi neku boju primetimo?

Trebaju nam objekti, nase oci i izvor svetlosti. Izvor svetlosti osveljava objekat, odbije se prema nama i nosi informaciju o boji. Ima nekolko parametara da simuilirana refleksija uradi posao kako treba!

- Pozicija svetla: nije svejedno dal je blizu, daleko, frontalno ili iza ledja (nece se nista videti ako je tu).
- Dolazni pravac (gde dolazi, gde gleda): ako imamo idealan tackasti izvor koji osvetljava na sve strane radijalno prikaz je korektan, slika je ostra, ali se detalji ne vide kako treba jer nema dovoljno svetlosti. Zato je takav tackasti izvor usmeren da gleda najvise u jednom pravcu, pa se dobije spotlight, gde sada vise detalja vidimo i ivice su ostrije (vidimo spotlight, tj. krug delovanja svih zrakova)
- Tip: ambijentalan (svetlo koje je uvek prisutno -> nije bitna pozicija i gde gleda jer je svuda prisutna istom jacinom, tackast izvor iz velike udaljenosti (Sunce), svetlost dolazi paralelno, daje nam informaciju dal je objekat tu ili ne, nije dovojno jako da nam da informaciju o objektu, kao sto je oblik, ili boja i materijal koje uopste ne prepoznaje!), tackast (prepoznaje boju jer ima dovoljno energije), spotlight (reflektorski, ima jos vise energije nego tackast)
- Intenzitet
- Boja: na nasem kursu izvor svetlosti uvek ce biti bele boje (emituje fotone svih frekvencija odredjene jacine)

Objekat: Svaki objekat ima neki koeficient refleksije. Objekat koji 100% moze da reflektuje svetlost je ogledalo. Kako se svetlost reflektuje? Pod istim uglom kod kojim je i doslo. Sto je ogledalo manje kvalitetno, neki procenat ce se reflektovani, a neki ce se apsorbovati. Oni sto se apsorbuju mogu na nekoj dubini da se reflektuju cim naidju na neku promenu gustine. Reflektovace se, odnostno apsorbovace se dalje. Mi radimo sa samo sa reflektovanom svetloscu!

Posmatrac: Interesuje nas samo pozicija.

Ako imamo neku tacku u koju kamera gleda pod nekim uglom na normalu, a pod tim uglom je usmeren i spotlight toliki ce biti intenzitet refleksije, toliko energije ce biti emitovana jer se posmatrac (mislim kamera) nasao pod istim uglom pod kojim i svetlo dolazi, a materijal je pravilan po geometriji (uglacan). Videce se bela tacka, a kako idemo dalje cemo videti boju. Ovaj dogadjaj zove se **spekularna refleksija**, jer toliki intenzizitet energije emituje da mi samo vidimo svetlo (ne vidimo nista). Ako materijal nije uglacan (hrapav je mikroskopski), onda svaka od malih povrsinica na njemu reflektuje svetlo pod svojom normalom kako je upalo i kad se to sastavi sve u stvari reflektuje sa svih strana. Ovo se zove **difuziona refleksija**. Ambijentalna refleksija daje samo oblik! Ovo je **Phongov** iluminacioni model!

Iluminacija vs sencenje (shading) : Nije, isto jer je sencenje aproksimacija prave iluminacije, a iluminacija je kada mi sracunavamo u grafici, kolko je objekat u nekoj tacki osvetljen i kolko se sa te tacke svetlost reflektuje. To radimo za sve tacke objekta! Za ovo bi nam trebalo jako puno procesne moci i zato se to ne radi nego se senci. Sencenjem zaprvo izaberemo jedan deo povrsine objekta i u jednoj tacki pronadjemo iluminaciju i taj deo povrsine ofarbamo sa istom iluminacijom! Sencenje je zapravo simulacija (aproksimacija) iluminacije na vecoj povrsini, ako ne radimo za svaku tacku objekta, nego je interpoliramo sa nekoliko tacaka (1,2,....,n). Sencenje ce nestati kada nam racunari budu tako mocni da mozemo da simuliramo na novou fotona pravu refleksiju. Ta povrsina za koju racunamo iluminaciju zove se **fragment**.

*Gledamo samo refleksiju sa objekta (kako mi dozivljavamo objekat osvetljenim).

*Materijal koji je malo uglacan, malo nije uglacan imace malo spekularne i malo difuzne refleksije.

*Mi igrajuci se sa parametrima zaprvo docaravamo korisniku vrste refleksije od materijala. **Iluminacioni modeli** su zaduzeni da na osnovu tih parametara proracunaju boju u tackama. Posle toga razmisljamo o modelu sencenja koji ce koristiti taj iluminacioni model. Iluminacija moze bez modela sencenja ali ce trebati jako jak hardver i jako puno vremena, a sencenje ne moze bez iluminacije.

*Sto se spekularna refleksija pojacava mi dobijamo vise informacija o tom objektu (ne menjajuci polozaj).

Phong-ov Iluminacioni model

Pronalaziiluminaciju neke tacke, tj. kolko se reflektovalo svetlo te tacke. Reflektovana svetlost neke tacke je suma ambijentalne komponente (uvek je tu) i reflektovana svetlost koja dolazi iz tackastog izvora (ne mora da bude tu uvek). Tackasti izvor nam daje informaciju o obliku i boji i u zavisnosti od njegovog intenziteta daje informaciju o materijalu! Ako imamo vise izvora svetlosti, za svaki moramo da racunamo kolko on doprinosi spekularnoj i difuznoj refleksili u toj tacki, a ako ne govorimo o beloj boji samo onda mozemo u komponenti boja (uzmemo odgovarajucu komponentu i racunamo).

Pitanje: Kako mi mozemo da uticemo na kolicinu ambijentalne svetlosti u nekom prostoru? Odgovor: Farbacemo zidove svetlijim bojama pa kad svetlost udje, dodatno ce se odbijati i povecavati intenzitet. Sto zidove ofarbamo tamnijim svetlostima apsorpcija ce biti veca pa ce intenzitet biti manji.

***Ambijentalna refleksija**:
Kad govorimo od cega zavisi ambijentalna refleksija, normala nema tu puno smisla jer svetlost “napada” objekatsa svih strana. Zato je refleksija odredjena intenzitetom upadne svetlosti i karakteristikama materijala(oni koeficienti za simulaciju). Ovde se ne treba “gurati” kosinus ugla.

2 pristupa refleksije ambijentalne svetlosti:

- Ambijentalna komponenta nosi korektnu informaciju o boji objekta
  Mozemo se igrati koeficientima za svaku RGB komponentu ambijetnalne svetlosti I tako dobiti boju objekta.
- Ambijentalna komponenta ne nosi korektnu informaciju o boji objekta
  Ovo gore pada u vodu. Smatra se da amb. svetlo nije dovoljnog intenziteta da mi vidimo boju.
  Primer : Sumrak! Sto vise pada mrak to mi vise I vise gubimo informaciju o boji objekata. Zato ona sama za sebe nije dovoljno jaka da donese boju, vec samo daje informaciju o oblicima.

***Difuziona refleksija**:
Difuziona refleksija zavisi od intenziteta upadne svetlosti, karakteristika materijala(koeficient refleksije za difuzionu refleksiju, to nam govori da li je objekat hrapav ili nalik ogledalu), kosinusa upadnog ugla (**θ**, ugao izmedju upadnog zraka I normale). Intenzitet tackastog izvora se moze dekomponovati po bojama.

Primer : Slika (slajd 110). Ako imamo samo ambijentalnu vidimo sve oblike. Ne mozemo reci da li je na slici slika, prozor

ili ogledalo. Difuzna ako je prisutna sama ce nam docarati da ono jeste ogledalo, ali se odraz zeca u njemu nece dobro videti. Kada se ove dve komponente spoje, dobijemo prilicno dobro izrenderovanu scenu, tj vidimo da se zec ogleda u ogledalu.

*Spekularna refleksija:
Kada se uglacanost materijala uskladi sa uglom gledanja (tj. taj ugao postane sve blizi uglu reflektovane svetlosti sa te tacke) mi dozivljavamo veliku energiju svetlosti iz tog pravca, da deluje da gubimo pojam o osnovnoj boji objekta (jer toliko puno svetlosne energije izlazi iz tih tacaka). Desava se kada nas ugao gledanja gleda direktno u odbijeni zrak sa povrsine (pod istim je uglom).
Intenzitet spekularne komponente zavisi od kosinusa ugla izmedju "At" vektora kamere I reflektovanog vektora svetlosti.To je ugao α (alfa). Koeficient (stepen) n simulira otvor/velicinu belila koje se pojavljuje. Sto je n (simulira kolicinu belila koja se pojavljuje) veci povrsinica je manja, ali je intenzitet spek. refleksije veci I granica povrsinice je ostrija. Obrnuto, ako je n manje, povrsinica je veca, intenzitet spek. refl je manji I granica je vise zamucena.
Zavisi I od materijala, ali I od geometrije objekta.

Kad se sve sabere, ukupno imamo 9 koeficienata kojima mozemo upravljati. Ne smemo zaboraviti da uracunamo slabljenje tackastih izvora (nikako ambijentalne svetlosti jer je toliko jako da nam to slabljenje malo koje bi bilo mozda prisutno nista ne znaci, tj. ne mozemo nikad tako daleko otici od objekta da dozivimo njeno slabljenje)! To isto treba simulirati.
Slabljenje cemo smatrati da opada sa kvadratom rastojanja (po fizici). Kada je kamera blizu i napravi mali pomeraj, tj. udalji se od objekta dalje, onda je slabljenje s ovom formulom mnogo vece. Ako odemo dalje od kamere i napravimo isti pomeraj, slabljenje u tom slucaju nece biti tako veliko. Odustalo se od toga ubrzo, ubacena je formula $\min(\frac{c1}{c1+c2*d+c3*d}, 1)$, gde su dodati koeficienti linearnog I kvadratnog slabljenja. Tako su dodata jos 3 parametra kojima se igramo intenzitetom upadne svatlosti. Imamo ukupno 12 koeficienata koje mozemo podesavati. Neki se odnose na materijal, neki na karakteristike svetlosti.

*Difuzna i spekularna refleksija uvek se pojavljuju zajedno, samo u udredjenim kolicinama u zavisnosti od materijala.

*Ako je objekat uglacan onda ce spekularna biti dominantnija, a ako je materijal hrapav onda je obrnuto.

*U ovom iluminacionom modelu mi samo pricamo o refleksiji svetlosti (a u realnosti se svetlost i apsorbuje)!

*Kada bismo uzeli boju u obzir onda bismo primenili istu jednacinu, ali za svaku komponentu boje.

Sencenje

Aproksimiramo/simuliramo na vecoj povrsini osvatljaj reflektovane svetlosti na osnovu manje informacija (ne radimo a svaku tacku, tada bismo skoro radili raytracing) kako bismo bili brzi. Algoritmi (modeli sencenja):

1. **Konstantno** : Svaka povrsinica se boji na osnovu jednog poziva iluminacionog modela. To znaci da se u jednoj (nekoj) tacki poligona na osnovu normale na osnovu neke matematicke formule(npr. ona Phong-ova od malo pre) sracuna osvetljaj. Poziva iluminacioni model jednom po poligonu. Na osnovu tog izracunatog osvetljaja jedne tacke poligona, bojimo ceo poligon. Zato se zove konstantno/homogeo, jer na osnovu iluminacije jedne tacke poligona mi bojimo ceo poligon. Lose je sto ako osvetlimo jednu povrsinu pomocu jedne tacke, drugu pomocu druge, razlika izmedju povrsinica bice previse velike (a mi zelimo da prelazi s jedne povrsine na drugu budu blazi).
   Ovo vise niko ne radi (zadnji koji su radili su mobilni telefoni koji nisu imali jaku procesnu moc), jako ruzno, ali jako brzo.
2. **Gouraud-ovo** (Guroovo) : Svaki poligon ima jednu normalu za celu povrsinu. Svako teme poligona je deljeno izmedju vise (bar 3) poligona. Zato trebamo usrednjiti normale u temenima I tako cemo ublaziti razlike u boji izmedju poligona.Po poligonu se iluminacioni model poziva onoliko puta koliko temena ima (uvek 3, radimo sa trouglovima). Od temena do temena cemo polako bojiti poligon u prelazima boje, interpolacijom!

Ovde se interpolira **BOJA** kroz citav poligon (Odgovor na pitanje kako radi gouraud-ovo sencenje!), moze biti problematicno. Bolje je mnogo od konstantnog, ali posto interpolira boju izgubi informaciju o normali vec na nivou temena. Za proracun spekularne refleksije nam je jako vazna ta normala, zato ako se spekularna refleksija javlja na nivou samo jednog trougla on je nece prikazati!Ona svejedno cesce zahvata vise temena I u tom slucaju ce biti uhvacena, ali nece biti adekvatno prikazana (ili ce preterati, sto je cesce ili ce je ublaziti).

3. **Phong-ovo** : Koristi interpolaciju normale po poligonu I za svaku interpoliranu normalu je pozivao iluminacioni model.Phong-ov model “popravlja” Gouard-ov tako da bolje prikazuje spekularnu refleksiju. Placamo time sto vise putapozivamo iluminacioni model po svakom poligonu, prenos podataka je veci, trosi se vise vremena, ali ce svaka spekularna refleksija biti uhvacena.
   Ide se opet po scan linijama, pronadje interpolirane normale izmedju temena I interpolira normale kroz svaki fragment (ovde je fragment skup poligona sa istom normalom!) scan-linije. Za svaku interpoliranu normalu se poziva iluminacioni model I sracunava se boja. Koliko ima fragmenata u trouglu, toliko ce biti interpoliranih normala, a samim tim ce se toliko puta pozvati iluminacioni model.

*Model sencenja != Iluminacioni model. Svi koriste Phong-ov iluminacioni model, ali ne moraju koristiti I njegov model sencenja.

***Raspodela posla kod shader-a**:
- **Gouraud** : Vertex shader usrednjava normale, cim on usrednji normale svako teme ima samo jednu normalu (usteda),ali moze da pozove ilum. model I nadje boju u temenima (tj. kako se reflektuje svetlost sa temena za koji je pozvan)! Prema tome, Pixel shader onda samo interpolira boju. Linearno interpolira (na osnovu sracunavanjausrednjenih normala i boja u temenima) boju po scan-liniji (red piksela) (slika 114)! Ranijim procesorima je to veoma odgovaralo jer je od vertex do pixel shadera putovalo manje podataka.

- **Phong** : Vertex shader usrednjava normale, ali ne sracunava boju. Svi koeficienti I informacije o materijalu idu do raster shader-a. Raster shader sada interpolira normale, pa na nivou fragmenta poziva iluminacioni model. Ima vise posla, ali je prikaz lepsi.

*Problem sa sracunavanjem normala u temenima:
Sencenje krova. Ako sracunavamo normalu u temenima kao na slici, dobicemo da su sve normale medjusobno paralelne(crvene strelice). To znaci da ce svi poligoni biti obojeni istom bojom, ispasce kao da je ravan potpuno. Resenje je da kod ovakvih (opozicionih) vektora ne trazimo bas usrednjenu normalu bas u samom temenu nego pridruzuju, tj. traze prvo teme do njega i usrednjavaju u tom temenu. Drugacije receno kao da to teme koje bismo inace trebali da usrednjimo malo gurnemo na jedan i malo na drugi poligon i tamo trazili usrednjavanje normala da bismo zadrzali te razlike u boji! Posto se ovaj problem desava i kod Gouraud-a i kod Phong-a moze se reci kao da se malo (ne previse) "vratimo" na konstantno sencenje.

Rasterizacija

Dolaze temena (sa osrednjenim normalama i bojom), mora se obojiti povrsinica izmedju tih temena. Nalaze se fragmenti, pikseli I bojimo ih. Snalazimo se kako nesto sto realisticnije obojiti, a da manje algoritamski platimo.

*Teksturisanje sa Z buffer-om je bio prvi posao koji je automatizovala prva generacija grafickh procesora.

***Rasterizacija poligona**:
Mora da pronadje svaki interijer I granicu svakog poligona kakav god on bio. Poligon je definisan kao zatvoreni skup linija, tj. prva definisana tacka prve definisane linije je poslednja tacka poslednje definisane linije! Redosled definisanja linija je proizvoljan, a algoritam mora da radi kako treba nezavisno od redosleda navodjenja. Poligon moze biti konveksan(svi uglovi sa strane interijera je $< 180^o$, ili ako spojimo bilo koje 2 tacke tako nastala linija ce ostati u

unutrasnjosti, lepog su oblika i sva ispitivanja i rad unutar njih je jednostavan) ili konkavan(obrnuto). Rasterizacija se moze obaviti:

- **"As-is"** : Bez isparcavanja na manje delove, koliko god temena imali. Naziva se scan line conversion algoritmom, jer ide po scan linijama, koje odgovaraju prikaznom uredjaju i trazi tacke enterijera. Zahtevan je i ima puno specijalnih slucajeva.
  Moramo obojiti enterijer I granicu poligona. Ide se po scan linijama I trazi presek sa ivicama I tako nalazi granicu poligona. Kako dobiti ivicu kada imamo samo temena? Jednacina prave kroz 2 tacke, scan linija je uvek paralelna sa x-osom (x je uvek 0, y ima neku odredjenu vrednost).
  Nacin obilaska zavisi od redosleda navodjenja. Zato nam je potreban prvi prolaz da pronadjemo sve preseke, sortiramo preseke po rastucem x-ui u drugom prolazu bojimo.
  Kako da znamo kako bojimo(konkavnost I konveksnost)? Krecemo uvek da bojimo iz eksterijera (necemo bojati do prvog preseka, nego bojimo od prvog do drugog). Imamo posebnu promenjivu (pocinje kao false posto krece iz eksterijera!) koja imitira semafor, koja regulise kada da se boji, a kada ne. Kako naidjemo presecnu tacku, prestanemo da bojimo dokle god ne naidjemo na anrednu presecnu tacku.
  - Problem1: Sta ako je presecna tacka teme i mi bismo trebali da nastavimo s bojenjem, a semafor bi rekao da prestane da se boji? Ako je to teme minimalno (konkavan) za obe svoje linije koje ga definisu nemoj uticati na promenu, tj. boji dalje, a ako nije maksimalna (konveksan) utici na promenu.
  - Problem2: Sta ako presecnu tacku dobijemo kao razlomljen broj? Ako idemo iz enterijera zaokruzujemo na manje,a ako idemo iz eksterijera zaokruzujemo na vece, kako bi razlika izmedju 2 jaako bliska poligona bila uocljiva!

Triangulacija poligona : Deljenjem jednostavnih poligona(ne ukrstaju mu se nesusedne stranice)na trouglove, ma kakav on bio (cak I sa rupom). Znaci ako dodje neki poligon koji nije trougao, pretvorimo u skup trouglova i rasterizujemo po trouglovima. Danas se vise ide u ovom smeru! Oslanjamo se na dijalonalu trougla (spoj dva nesusedna temena i svaka tacka te dijagonale mora biti element enterijera!). Samim tim nas posao je da dijagonalama i stranicama tog poligona formiramo trouglove unutar njega (npr. dve dijagonale i stranica ili dve stranice i jedna dijagonala!). Iskljucuje kompleksne situacije kada se dve nesusedne stranice seku jedna drugu sto moze napraviti problem. Nece biti jednoznacnog resenja, tj. moze na vise nacina da se izdeli neki poligon, a nama je bitno da bude sto brze!
*O(n^2) slozenost: Najgori slucaj je da ce se jedno izvrsavanje algoritma sto mi vise povecavamo broj ulaza, srazmerno tome ce njihovom kvadratu trebati da algoritam nadje resenje.

- **Ear clipping**: Treba naci glavno teme (njegovi prethodni i naredni sused (teme) formiraju pravac koji sece poligon samo u njima, tj. ne u jos u nekom drugom temenu ili tacki tog poligona i pripada enterijeru trougla -> prava dijagonala). Ako konveksno teme nije glavno onda se u ondnosu na njega nadje reflektivno teme tako sto se bira teme koje je najudaljenije od 'dijagonale'. Onda se ono poveze sa svim temenima koja su mu susedna i sa onim cije je reflektivno i onda mozemo da podelimo poligon na trouglove.

- **GJPT**: Podeliti poligon na trapezoide, pa njih na monotone delove (sastavljeni su od dva lanca ivica poligona, ali ta dva lanca moraju biti monotoni, a monoton lanac je onaj koji u odnosu sanormalom na x-osu ima samo jedan presek sa tom osom). Dalje te monotone delove podeliti na trouglove. Na trapezoide delimo sweep algoritmom tako sto na ortogonalno na x-osu napravimo pravac i onda za svako teme poligona povucemo pravu na suprotnu ivicu.
  Trapezoide delimo na monotone delove tako sto se otkrivaju **reflektivna temena** koja koja mogu biti **split (iz tog temena izlaze (rastavljaju se) dve ivice)** ili **merge (u tom temenu se spajaju dve ivice)** tipa. Dal je teme split ili merge gledamo u odnosu na rastucu x-osu! Dalje ako dolazimo do reflektivnog temena i koje je split tipa, izaberemo njemu prvo levo teme u njegovom trapezu i spojimo njega i njegovo prvo levo teme (dobijamo jedan deo poligona koje mora da je monotono). A ako je merge tipa onda se gleda prvo desno teme i s njim se spaja.Trouglovi monotonih delova se traze tako sto se ide redom i prati normala na x-osu i gleda se dal temena imaju dijagonalu, ako ne ide se dalje dok se ne nadje i ako je potrebno rekurzivno se vraca da bi vrsio korekciju (tako sto gura na stek oni s kojima nije formirana dijagonala)! Ima dosta posebnih slucajeva!

Kako rasterizovati trouglove?

- Delimo prostor i deljenjem prostora docicemo do elementarnog dela prostora, koji ce biti fragment (koji kasnije postane piksel) ili piksel.
  - **Rekurzivna podela na mikropoligone** : Delimo enterijer. Svaki trougao delimo na manje trouglove tako sto nadjemo sredinu stranica, spojimo ih po dijagonalama (tumaci kao nova temena) i tako dobijemo 4 nova trougla.

  - **Warnock-ov algoritam** : Delimo citav prostor sve dok ne dodjemo do nekoga ko je homogen po boji (fragment). Kako ubrzati? Oko trougla opisemo jedan ekran (kvadrat) i onda brze konvergiramo ka resenju.

  - **Edge equation algoritam** : Trougao se sastoji iz ivica. Svaka ivica je duz opisana svojom jednacinom (jednacinom prave). Svaka duz (ako je predstavimo kao pravu) deli prostor na 2 dela podprostora. Svaki od podprostora ima svoje neke karakteristike. U nasem slucaju svaka prava bi delila prostor na 2 podprostora koji bi medjusobno imali iste karakteristike (Slajd (9), slika). Zato idemo od piksela do piksela I gledamo gde se on nalazi u odnosu na sve 3 prave. Ako je za sve 3 u pozitivnom delu, u unutrasnjosti je! Ako je za bar jednu neispunjeno, ne moramo dalje ispitivati, sigurno je napolju!
    Sta nam govori sa koje je strane? Pomocu jednacine prave. Imamo koordinate temena i iz vektorskog mnozenja mozemo dobiti koeficiente A, B I C I tako dobiti jednaciju prave. Onda za bilo koju (proizvoljnu) tacku mozemo izracunati vrednost $Ax + Bx + C$ I ako je ta vrednost >0, tacka je u pozitivnom delu u odnosu na tu pravu, ako je <0 u negativnom je. Ukoliko je tacka u unutrasnjosti ova nejednakost ce se ispitati 3 puta (tj. po jednom za svaki jednacinu prave), a ako je spolja sve dok se prvi uslov ne ispuni.
    Ponovo moguce ubrzati opusuvanjem cetvorougla oko trougla za ispitivanje.

- Idemo po scan linijama i radimo scan line algoritam koji je specijalizovan za trouglove i ima manje specijalnih slucajeva.
  Edge walking algoritam : idemo po scan linijama i posto znamo da je trougao neko teme mora biti gore, neko dole i jos imamo srednje teme koje je ili sa leve ili sa desne strane. Krece od gornjeg i ide ka donjem. Ako je srednje teme sa leve strane onda ce na njega smanjivati x, a prema drugom temenu povecavati na x. Kada dodje do temena na koje je smanjivao x morace da to smanjivanje promeni u povecavanje x-a i prati (hoda) po liniji i izmedju dve tacke svake linije treba da boji.Ako nema breakpoint (tacku loma gde okrece x), to znaci da je ili jedna stranica paralelna x-osi I moze se obrnuti y-osa.
  Ovde se prvo razmisljalo sta cemo sa pikselima koji pripadaju I enterijeru I ivici! Resenje je **aliansing**, gde mi namerno razmazujemo piksele tako da izgleda kao da je ivica ravna. **Jaggies** su stepenice i njih uvek imamo u rasterskom prikazu, sa povecanjem rezolucije stepenice su manje, ali je i linija tanja, tj. i zato koristimo aliansing da to izbegnemo. Najosnovniji algoritam za crtanje linije boji liniju onom kolicinom boje kolko je idealna linija prekrila taj piksel. Ako je piksel prekriven oko 100% njega bojimo sa bojom linije, sve ostale piksele bojimo bojom koja je jednim procentom kojim je prekriven idealnom linijom, a drugim procentom boja pozadine.

Aliansing:Prvo vidjena kod klasicnih monitora i televizora. Ako je snop elektrona koji putuje ka fosfornom zaslonu, ako dobro podesen (ide samo horizontalno) onda ce "prvi" elektroni ce malo uraditi s leve strane sredine, najveci broj ce udariti u srednju tacku gde i treba, a oni sto kasne ce udariti malo s desne strane. Kao rezultat svetlece najvise tacka ona sto treba, ali ce malo svetliti i ona susedna (tj. tacka se razlila, nije dovoljno fina). Zato je napravljena anti-aliansing mreza gde je iznad svake tacke (fosfora, tj. fizickog piksela na ekranu) stavljena celicna ploca na kojoj su izbusene rupe iznad svake tacke. Ta mreza ode u pozitivan napon sve dok ne dobijemo u najostriju sliku, jer pocetni i zadnji ekstremi snopa nece stici do prethodnih i narednih tacaka jer ce ih pozitivan napon privuci na sebe. On ce i povuci i one elektrone sto su trebali da stignu na sredinu, ali je to zanemarljivo tj. nije tolko jako da ugasi tu tackicu. Samim tim slika je izostrena, jer je ova mreza sprecavala razmazivanje tacaka.

*LOD : Govori kolko imamo trouglova. Prilagodjava se ekranu. Ako neki objekat ode od kamere on mora da se smanji (manji LOD) u odnosu kada je blizu kamere. Tako da kada se modeluje objekat se pravi u nekoliko LOD-ova, napred najfiniji i onda se smanuje po npr. 1/2, 2/3 i 1/3 toga u zavisnosti od pozicije objekta.

Pix-maps : 2D mapa piksela (matrica), ima redove I kolone, to je u stvari rezolucija. Sto slika ima vecu rezoluciju ona je finija, lepse su tacke, mozemo vise detalja staviti na nju (bitno je staviti dovoljno detalja). Koliko boja ima na slici, to je dubina (depth), tj. kolko je bitova zaduzeno za opisivanje boje na toj slici. Ukoliko je dubina 1 (1 bit se koristi za opisivanje boje piksela), slika je iskljucivo sa crnim I belim pikselima (bukvalno **black and white**). Sve preko jednog bit-a je zaprvo neka boja, pri cemu ta boja moze biti po nekoj skali, npr. od crne do bele, sto bi bila **slika sa novoima sivog (greyscale)**. Danasnja dubina je 24 bita (po 8 bitova za svaku komponentu RGB, a moguce je i dodati jos 8 bitova za alfa komponentu koji definise kolko je nesto **transparentno**). S njima mozemo raditi svasta: rotiranje, algebarske operacije, kombinovanje (12 operacije), igranje sa alfa kanalom (da nestane i da se pojavi) i skaliranje (promena dimenzija). OPENGL je prihvata i salje odmah u fazu rasterizacije.

Skaliranje : Za povecavanje i za smanjivanje rezolucije. Smanjivanje je jednostavnije jer mi od vise piksela pravimo manje. Skaliranje na manje **gubi detalje** sigurno, radi se tako sto mi trazimo medijanu, tj. srednju vrednost od nekih boja koje su u rasponu. Skaliranje na vece radi se **interpolacijom**, tj. pravimo novu boju kojoj dodeljujemo vrednost koja mora biti vrednost izmedju postojecih starih kako bi prelazi bili lepsi. Interpolacija mora biti **bilinearna** (posto ide po dve dimenzije, a ne linearna koja bi isla po jednoj dimenziji). Linearno skaliranje bilo bi ili samo po vertikali ili samo po horizontali i to bi bilo destruktivno za sliku (sadrzaj se menja)!
Prvi nacin (**nearest neighbor**) je da nove piksele napravimo tako sto cemo ponoviti boju najblizeg suseda (npr. jedan piksel se poveca na 4 i sva ta 4 imaju tu originalnu boju), lako i brzo je, ali je ruzno.
Drugi nacin (**bilinearna interpolacija**) je da bojimo nove piksele sa prelazima boja. Idemo po 2 ose, radimo linearnu interpolaciju prvo po horizontali, pa te interpolirane vrednosti koristimo za interpolaciju I po vertikali. Moze I po vertikali pa horizontali, svejedno, ali neka mora prva. Problem je u tome sto se mogu naci veliki prelazi kada radimo interpolaciju izmedju neka 4 piksela I onda 2 od ta 4 koristimo I za narednu interpolaciju.
Npr : ABCD, izvrsimo interpolaciju, predjemo na interpolaciju CDEF. Iako su C I D deljeni izmedju obe grupe, izmedju novonastalih piksela se mogu javiti velike razlike! Distorzija na malom lokalu.
Taj problem resava **bikubna interpolacija** tako sto piksel bojimo tako sto uzmemo veci procenat uticaja pravog suseda i manji procenat uticaja suseda po dijagonali**.** Lepsa je, ali je teze matematicki izracunati i treba veca procesna moc. *Sto je sused dalji to je njegov uticaj na interpolaciju manji!

Teksture

1. **2D** : Prevlacenje dvodimenzione slike preko trodimenzionog objekta.
2. **3D** : Definisemo materijal I kao klesar iz njega izdvojimo nas objekat. Materijal se definise algoritamski, proceduralno, I to su ustvari vrednosti boje **vokseva** (volume pixels), a onda se nad njima definise koji su voksevi od objekta a koji nisu. Ovde nemamo samo teksturu na spoljasnjosti, vec ako razlupamo objekat videcemo da tekstura u unutrasnjosti prati pojave na spoljasnjosti.

***Tehnike 2D teksturisanja**:

1. Crtanje po objektu : Radimo sami, rucno, obicno se koristi za detaljisanje.
2. Environmental (raflection) mapping : Mapiranje okoline. Na taj objekat se lepi refleksija spoljasnjosti (docaravano da je taj objekat jako reflektivan, da je napravljen od takvog materijala u kome se reflektuje okolina). Imamo objekat i sve sto se oslikava od tog objekta, tj. njegove okoline najcesce se mapira na sferu jer ona najmanje vrsi distorziju u prostoru i tu sliku onda nalepimo na taj objekat.
   Primer 1 : Vrlo sjajna sfera, vrtimo kameru oko te sfere I od snimka je napravljena tekstura koja se moze nalepiti na neki drugi takav objekat. Na spojevima se moze javiti distorzija.
   Primer 2 : Terminator, jak odsjaj lica coveka na kacigi. To je inace snimljeno u normalnom ogledalu, ali je iskorisceno da se prevuce preko modela te kacige I da efekat velike uglacanosti.

3. Displacement: Promene (modifikacije) nad geometrijom.
   Primer : Pravljenje narandze od lopte. Temena lopte blago izvlacimo I udubljujemo da bismo dobili grbe.
4. Bump mapping : Ne menjamo geometriju, vec manipulisemo normalama . Tako pomerene (modifikovane u odnosu na originalne u nekoj proceduri) normale saljemo shader-u na prikaz. Nakon renderovanja daje utisak kao da smo mu menjali geometriju, jer je zbog promenjenih normala sencenje drugacije. Jeftin trik.
5. Proceduralno mapiranje teksture: Algoritamski je definisano sta se na cega mapira I prenosi. Imamo 2D sliku I delove te 2D slike lepimo na delove (trouglove) objekta, gde svaki dobije neko parce slike. Slika ima svoj koordinatni sistem (obicno s i t koordinate da bi se razlikovalo od x i y ako je u svetu rasterizacije ili od x,y i z ako je u svetu world kordinata). Tekstura se sastoji iz teksela (texture element, piksel teksture). Mapiramo teksel na verteks (teksel t1 odgovara vertexu v1, itd.). Za vreme renderovanja kako rasterizujemo (npr. trougao) kako nadjemo element interpolacijom pronalazimo koji teksel njemu odgovara, uzimamo njegovu boju, dodajemo elementu i tako prevlacimo teksturu na taj trougao. Mapiranje je moguce izvrsiti direktno, odmotavanjem 3D modela, tj. njegove ravni i nadjemo poklapanje sa teksturom ili koriscenjem nekog pomocnog objekta(obicno neka pravila tela poput kocke (ako je objekat na koji lepimo kockast), lopta, cilindar… posto se na njih lakse lepi tekstura). Teksturu mapiramo na taj pomocni objekat (medjuobjekat) i onda s tog objekta mapiramo na onaj sto smo zeleli. Moze raditi I sa 3D teksturama.
   Problem moze biti u odrzavanju kontinualnosti, tj. da odrzimo lepe I neprimetne prelaze.
   Primer : Na model zeca prevlacimo teksturu aligatora.

***Problemi**:

- Odrzavanje kontinualnosti : Odrzavanje lepih I neprimetnih prelaza.
  Resava se “slikarski”. Kada trebamo neku veliku sliku naslikati na zidu, zgodno je projektovati osnovne linije, njih prvo iscrtati, pa popuniti ostatak. Tu filozofiju su preneli I ovde. Iz neke tacke vrsi se planarna projekcija na objekat. Ne sa objekta na ravan kao do sada, vec sa ravni na objekat!
- Razlike u mapiranju : Ukoliko imamo situaciju da nam je objekat u pokretu I priblizava se I udaljava od scene. Ako smo ostro vezali teksel za verteks prilikom uvecavanja I smanjivanja doslo bi do udaljavanja/priblizavanja verteksa tog objekta. Ispasce da jedan verteks interpolira i nalazi svoju sliku na vise teksela na teksturi(udaljavanje) ili nam mogu nedostajati tekseli (priblizavanje).
  Resava se posredovanjem : Mapiranje teksture na madju-objekat. Sto je taj objekat blizi po obliku onom na koga stvarno lepimo teksturu, spajanje ce biti lepse. Npr. ako je cilindar posrednik da li lepimo normalno sa njega na objekat ili da nam pravac kretanja diktira povrsinica, pa da idemo normalno na povrsinu prema cilindru ili da idemo sa centra ose tog objekta.

*Kako se menja polozaj objekta u odnosu na kameru, mora se menjati I njgova tekstura, tj. ona mora da prezivljava sve promene kroz koje prolazi i objekat od kada je definisana.

***Mipmapping**: Tekstura se unapred meorise u nekoliko pravilnih rezolucija, gde je svaka duplo manja od predhodne. Neko mora da vodi racuna da li se objekat udaljava ili priblizava, bira se odgovarajuca tekstura I lepi na objekat.
Nema ponovnog racunanja I dodatne matematike, ali placamo time sto predprocesiranje mora biti vece. Trosimo vreme na bilinearni resize, memorisemo tih vise verzija tekstura I u vreme renderovanja se bira ona koja najvise odgovara. Obicno budu dobro uskladjene, pa ide teksel na piksel.

* **Tehnike 3D teksturisanja** :
Imamo matematiku koja ce nam nekom funkcijom (koju mozemo algoritamski podrzati) izgenerisati teksturu/matrijal na osnovu odredjenih parametara.
Osnovna funkcija jeste **Noise**(šum), kontinualna je, ide po sve 3 dimenzije prostora, generise slucajni broj koji ce odrediti boju vokseva na odredjenim pozicijama. Drugim recima ide kroz citav prostor i samo 'lupa' boje. Daje zrnastu strukturu, voks do voksa se mogu jako razlikovati po svetlu onoliko koliko nam je generator slucajnih brojeva dobar Sto je generator bolji, tj. pravi bolju slucajnu distribuciju brojeva, nama ce materijal biti zrnastiji, tj. bice veoma velike razlike od

vokseva do vokseva. Tom noise-u su dodali turbulenciju, kako bi se na nekim mestima nagomilavale tamnije vrednosti, a na nekim mestima ostavljale svetlije vrednosti praveci kojekakve matrijale. Podesavanjem parametara noise moze praviti “zile” po I u materijalu(primeri mermera I drveta sa slajda(22)). Mi sa odredjenim vrednostima i odredjenim brojem noise-a stvaramo nagomilavanja odredjene boje i stvaramo utisak da je to neki materijal. Dobro je sto je verodostojnost maksimalna sta god da uradimo s tim objektom (prepolovimo ga npr.).
Ograniceni smo sa brojem materijala koje znamo proceduralno izgenerisati.

Frame buffer

U sebi cuva opis boje svakog piksela. Postoji relacija izmedju adrese lokacije u frejm baferu I lokacije piksela na ekranu (0-ta adresa odgovara prvom px u prvom redu). Ta relacija je uspostavljena da bi se olaksao posao Scan-kontroelra, da on bude sto jednostavniji, a da da brz odgovor na ekranu.
Frame buffer ima 4 dela:

1. **Color buffer** : Cuva boju svakog piksela u bilo kom obliku. Prvo su postojali samo front I back za dvobafersko punjenje. Imali ranije, dok se sadrzaj jednog prikazuje, drugi se puni I onda se switchuju. Uvedeni su left I right, za 3D tehnologiju(stereoskopiju, da svako oko gleda svoju sliku i tako dobijemo osecaj da gledamo u 3D sliku).
2. **Debth buffer** : Cuva dubinu, tj koliko osvetliti tu neku boju
3. **Stencil buffer** : Cuva sablon matricu. Dobri primeri onaj kokosinjac, njihanje travice, oblaci (sabiramo tu zamucenost oblaka sa prikazom u color baferu, pa ispadne kao da je to zamuceno), voznja u automobilu…
4. **Accumulation buffer** : Buffer za “sve ostalo”. Uvek postoji neki bafer koji akomulira u sebi neke medjurezultate koji ce se kasnije spajati sa nekim drugim rezultatima ili sluziti kao prosirenje nekog od narojanih bafera.
   *Primeri: Moze cuvati rezultate renderovanja, moze ih sabirati da bi dobio razlicite efekte poput izostravanja blurovanja. Promena rezolucije slike(supersampling -> povecavanje rezolucije) da bi se eliminisali neke nedostatke… Ne mora biti koriscen uopste.

*Ekran vs Monitor : Ekran je ono u sta gledamo, a monitor je I kompletna masina iza(ceo uredjaj).

MDL

AutoDesk specifikacija za pravljenje grafickog softvera. Gura grafiku napred. Proizvodi : 3D Max, AutoCAD…
Omogucava da se lako isprogramira **geometrija** I pokret nad objektima (**animacija**). Koristi se objektno programiranje. Tj. ako izmodelujemo neki objekat I isprogramiramo(razvijemo) njegov pokret pomocu svih onih matrica transformacije, drugi objekat ga moze naslediti I dodati neke svoje specificnosti . Razvio se iz “gladi” za gejmingom, tj da ne tako mocni graficki hardveri I softveri mogu prikazati razlicite graf. prikaze I animacije.

*Primer 1: Razvijamo model konja. Izmodelujemo ga, isprogramiramo koracanje, skakanje, ritanje itd.. Tu specifikaciju konja je moguce postaviti na internet. Neko ko skine/kupi tu specifikaciju konja, moze je naslediti, dodati grbu, napraviti vezu te grbe sa kicmom konja I relativno lako dobiti kamilu.
*Primer 2 : Pingvin. Moguce je povezati vetrekse i teksele (teksel t1 odgovara vertexu v1, itd.) i tako dodeliti teksturu objektu.
*Primer 3 : Model I animacija jelena. Citav objekat (geometrija) jelena predstavljen strukturom stabla. Svaki deo stabla (noga, trup, itd.) moze da se iskljuci u animaciji. Postoje matrice veze izmedju podredjenih I nadredjenih elemenata.

Uredjaji

*Pisaca masina : Klasicna mehanika. Mehanicko pritiskanje dugmadi, svako dugme odgovara jednom slovu sa lepeze na kojoj se nalaze negativna(flipovana) slova koja kada udare u mastiljavu traku(prvo crnu, kasnije I crvenu) I ostave trag na papiru. Prelezak u novi red se odvijao rucno, namotavanjem I navlacenjem poluge. Shift taster naredjuje da se nosac mastiljave trake podize i onda kucamo po crvenoj polovini, tj. dobijamo slova crvene boje.

*Racunovodstvena masina : Paskalov mehanicki sabirac. Pritiskali smo brojeve I okretanjem rucice su se postavljali zupcanici kako bi se taj broj “zapamtio”, isto tako I za drugi, izabere se operacija I izbacice se rezultat na traci. Kasnije se ugradio elektro motor, da ne mora mehanicki.

*Telegraf : Prenos poruke(slova) na daljinu. Koristio je telefonske linije, kasnije je dobio svoju liniju kojom se slao analogni signal koji je predstavljao poruku.

Sve sto je imao do sada, covek je “as-is” usvojio I u racunarstvo I usavrsavao. Nije od nule I nikako potpuno novo. Imali smo klasicne, stenografske tastature, miševe, touch-screen, detektori pokreta sake, virtual-reality, ide se ka tome da nema pritiskanja dugmica.

*Stenografska pisaca masina (orgulje): Nije jedan klik jedna cifra, nego jedan pritisak jedno slovo, drugi pritisak drugo slovo, itd. Ako koristimo 5 pristiju imamo 32 simbola (2^5). Brzina unosa je oko 300 reci u minuti i to je dobro za hvatanje zapisa (npr. u sudu).

*Detektor pokreta sake i prstiju: Prati sake i prstice kolko se savijaju i uvijaju pomocu rukavice. Na rukavicama ima puno namotanih zica (merna traka), kako mi savijamo prste tako ce se zice rastavljati (siriti) i menjati svoju indukciju i tako cemo detektovati naizmenicnu struju i kolko je korisnik savio ruku. Ovde je lose zato sto gledamo prst kao da je gumen, tj. da nemamo zglobove. Usavrsavanjem krenuli smo da pratimo i svaki zglob. Problem je sto ne mozemo korisniku da damo osecaj za pritisak.

Podela:

1. Tastature : Unos alfanumerika.
2. Pointerski uredjaji : Pozicioniranje i selekcija. Mis, joystick, touchpad, izometrijski joystick na laptopu…
3. Prikazni uredjaji: E-copy(ekran, projektor…) I H-copy(stampac)

Tastature

Sluze za unos alfanumerika. Alfa deo za unos karaktera i numericki deo za unos numerika. Raspored cifri na numerickim tastaturama preuzet sa starih fiskalnih kasa, koje su imale red cifara za svaku jedinicu (jedinice, desetice, stotine, hiljande...), gde su najvece cifre postavljene vise, da bi bile u visini pogleda prodavca da bi manje pogresio pri unosu tih velikih cifara.
Probali su da izmene raspored dugmica kod mobilnih (npr da se uvede kruzna organizacija), ali je I dalje ovaj prvi raspored pobedio, zasto? Nacin na koji mi gledamo. Od gore do dole I s leva na desno, ali I to da se prvo morala podici slusalica I kada je coveku vec pogled na njoj, lakse I brze bira broj.
Prebacivanje alfa dela. Najstariji je **QWERTY** raspored, stariji i od racunara(1870.), raspored isti kao na starim pisacim masinama. Organizovan je tako jer su se trudili da se podjednako podele na levu I desnu ruku najfrekventnija slova tog jezika na prste koji su najvise angazovani. Isto, kada su pisace masine u pitanju, moralo se paziti I na ogranicenja mehanickog dela, tj. kada izlaze slova u isto vreme da se ne zaglave.
Posto ovakav standard za srpski jezik ne postoji(nije sprovedena studija) I kod nas se desavalo da se cesto slova zaglavljuju.
***Vrste tastatura**:

1. Mehanicke : Fizicke, npr. tipicna za desktop racunar.
2. Elektronske : Na ekranu, podrazumevaju touch-sensitive ekran.

Neke nedostate QWERTY je popravio **Dvorak**, isto stariji od racunara(1920.). Brze kucanje (omogucava oko 150 reci u minuti), manje zaglavljivanja. Uzima samo frekvenciju pojavljivanja slova i da podjednako budu opterecene obe ruke, tj. prsti. Postojala je I **ABCDE** za apsolutne pocetnike (da se lako nadju slova), nema optimizacije.

***Optimizacija izgleda I ugodnosti koriscenja tastature** :
Razmisljalo se o obliku tastature, oblik i velicina tastera, kolikom silom da se pritiskaju tasteri, kako korisnik da oseti da jeste pritisnuo nesto(omoguciti tasteru da moze da “ulegne” u tastaturu), a da ne razvaljuje kao po pisacoj masini (da ne

cujemo klik, vec da osetimo klik)… Uvede se pokazivaci stanja (npr. diodica za Caps lock), oslonac za dlanove… Kod laptopova je potrebno voditi racuna gde smestiti unutrasnje komponente I cooler-e tako da ne greju povrsinu tastature, narocito taj oslonac za dlanove.

*Pitanje : Kako smestiti citavu tastaturu u laptop koji je debljine par centimetara i sve unutra je par milimetara, a da obezbedimo da se taster krece par milimetara, tj. kako odvojiti pola debljine laptopa za taster, a koristnik da ima dobar osecaj kucanja. Taster vise ne uleze, nego se “navije” u jednu stranu, prosto se klacka. Pa ga prst ne gura dole nego klizne niz taster i onda on ode milimetar untra, ali se i navije i onda jos 1, 1.5, 2 milimetara klizne prst i mi imamo osecaj kao da je uslo unutra.

*Pitanje : Zasto su gotovo svi integrisali pointerske uredjaje I tastaturu? Kako bi se eliminisao Homing!

***Fizicka vs Elektronska tastatura**:

- Fizicka daje bolji osecaj za kucanje I feedback ima. Kod elektricne se to popravlja vibracijom I zvukom, sada vise I vizuelnim efektima(npr. slovo na koje kliknemo se na kratko uveca). Kod ergonomske tastature je obavezno slepo kucanje! Zbog ergonomske, klasicne tastature prelomljene su na sredini i povecali razmak za 9 cm, da bi se poklapapo sa prirodnijim polozajem. Da bi nas naterali da dlanove stavimo na odmaraliste podigli su jos veci ugao tastature i to garantuje da se necemo umoriti.
- Elektronska ne zahteva nikakav svoj zasebni fizicki prostor za upotrebu(npr. radni sto), ali zauzima ekranski prostor. Ugradjenja je kod skoro svih hand-held uredjaja, pa je prednost sto se ne mora nositi dodatna tastatura, ali zauzima ekran. Zahteva touch screen ekran (nije vise tako drasticna cena, ali obican ektan + tastatura su I daje jeftiniji od jednog velikog touch-screen ekrana). Kod elektronske mozemo sami, dinamicki, da organizujemo layout slova, ali nevolja s tim je sto treba 'izdresirati' korisnika. Elektronska otvara mogucnost unosa slova(i reci) gestikulacijom! Tj mozemo prstom ili olovcicom napisati citavu rec, ne moramo da kucamo. To kod fizicke ne moze nikako. Moze biti omoguceno i prevlacenje kao unos (ispis “b-e-s-t” sa slajda)…
  Kod gestikulacije problem moze biti to sto dok kucamo prstima, ukoliko nemamo ciste ruke, ekran se moze zaprljati I tada bi se javilo preskakanje. Postoji I opcija prediktivne tastature! Tj. za pointiranje, povrsina targeta (slovo sa vecom verovatnocom da bude sledece) ce se povecati. Drugi problem je sto trebati obuciti koristnike za to.

*Pitanje : Kako covek cuje? Zvuk su promene pritiska u vazduhu, prvo imamo spoljasnje uho koje redirektuje zvuk ka bubnoj opni (deli spoljni svet od unutrasnjeg), onda imamo cekic, nakovanj i uzengija (pricvrscena u vezi s drugom malom opnom unutrasnjeg uha), prinesu pojacano pomeranje na unutrasnju bubnu opnu, a zidovima puza (unutra je tecnost) su dlacice (nervni zavrseci) i pumpa tecnost, mrdaju se dlacice, salje se impuls mozgu i mi to pretvaramo u zvuk. U unutrasnjem uhu imamo dva polukruga sa tecnoscu, polukrugovi imaju malo vazduha u sebi, pa kako okrecemo glavu tako se okrecu i parovi polukrugova i mi onda znamo gde smo glavom ili telom u odnosu na zemlju (osecaj ravnoteze). Kada mi stavimo slusalice, diretno napadajemo zvukom na bubnu opnu koja luduje. Cekic, nakovanj i uzengija povezani su malim misicima za suplje kosti (i preko njih se prenosi zvuk) i onda oni smanjuju pojacanje. Da to ne rade eksplodiralo bi nam unutrasnje uho. Ako mi jako dugo slusamo nesto, takvom masinom, nasi misici su stalno zgrceni i moze da zaboravi da se opusti, pa izgubimo pojacanje i ostanemo bez slusnih aparata.

Pointerski uredjaji

Zadatak im je da obezbede “point and click”. Pointiranje se vrsi nad objektima koje korisnik vidi, tj. prikazani su mu na ekranu. Pointerski uredjaji se mogu to obezbediti na 2 nacina:

- uredjaji **Direktne** kontrole (upravljivosti) : Primer prsti ili olovcica za touch screen, svetlosno pero. Imamo osecaj da direktno pritiskamo, tj. upravljamo sadrzajem. Direktno ga dodirujemo.
- uredjaji **Indirektne** kontrole : Primer mis, trackball, joystick... Indirektnost je u tome sto ga “vozamo” po stolu kako bismo pomerili pointerski kursor na ekranu. Ne vrsimo pointiranje uredjajem, vec postoji neka vrsta

posrednika (kursor). Drugim recima vizuelnim prikazom manipulisemo ne direktno, nego uredjaj koristimo tako da neki njegov posrednik dodje u poziciju u koju treba i tu nesto radi. Npr. misem pomeramo kursor da on ukaze na neki vizuelni element interesa.

Direktni vs Indirektni :

- Direktni : Nemao tako dobar feedback kao kod misa npr. (osetimo da smo kliknuli). Pointiranje prstom nikada ne gadjaju 1 px. Bitna prednost je u tome sto je moguce elegantnije I lepse crtati slobodnom pokretom (npr. crtanje necega u paint-u misem ne lici ni na sta). Najcesce ne zahtevaju nikakvu obuku za koriscenje. Ne koriste dodatni fizicki prostor samo prikazni uredjaj.
  Koriscenjem derektnih zaklanjamo onu stanu ekrana rukom koju koristimo! Koriscenje direktnih mnogo vise zamara korisnika, moras drzati ruku u vazduhu najcesce. Manja obuka je potrebna za koriscenje njih.

- Indirektni : Malim pomerajem indirektnog uredjaja vise mozemo da predjemo i brzinu tog pomeranja najcesce mozemo i podesiti. Postoji mentalna priprema pre koriscenja da pronadjemo gde nam je kursor i odakle vrsimo pomeranje. Potrebno je bar malo vezbati klik, dvoklik, pomeranje itd… , tj. za njih je potrebna obuka za koriscenje njih (koordinacija ruka i oka npr.).
  Zaklanjanje ekrana je onoliko kolika je velicina kursora (sto je veoma malo). Svi indirektni koriste neki deo fizickog prostora(podloga za misa), ali znacajno manje zamaraju korisnika prilikom koriscenja.Digitalna tabla za crtanje je isto indirektni, omogucuje crtanje slobodnom rukom, daleko udobnija za koriscenje nego da treba da crtamo direktno po ekranu.

*Pitanje : Razvijamo softver koji treba da prikzuje neke opste informacije o gradu (npr) koji pointer ubaciti?
Trebalo bi izbeci indirektni, jer ne zele ljudi da uzimaju u ruke nesto sto su svi dirali. Za direktni je manja povrsina dodira.

Uredjaji direktne kontrole :

***Svetlosno pero** : Direktni je, najstariji. Radi tako sto video kartica cita polozaj svetlosnog pera sa ekrana, proracunava kasnjenje I prikazuje polozaj na ekranu. Osetljivi na nivou tacke. Bilo je zamorno drzati ga, pa su ljudi osmislili taster za zadrzavanje pozicije. Reagovao je isprva na “land-on” (spustimo pa reaguje) pa su potom prebacili na “lift-off ” (uradimo land on i tek kad odmaknemo onda reaguje, sto sada rade svi pointerski uredjaji) kako bi se greska prilikom koriscenja umanjila.

***Ekran osetljiv na dodir (touch-sensitive)** : Rezistivni I Kapacitivni. Pristima pokazujemo.
Rezistivni su se prvi pojavili. Prevucen je folijom, koja je u stvari unija vise folija, zeleom sa jastucicem (velicina odredjuje kolko je osetljivo polje koje se pritiska) I kada se pritisne neki jastucic, on se blago deformise, menja mu se otporna karakteristika I to se detektuje kao dodir. Mora se malko jace pritisnuti i zbog materijala izrade (zelea) osvetljenje slabi, pa se mora pojacati osvetljenje i vise struje trosi. Dobri su za embeded sisteme i u industrijskim postrojenjima, jer su **otporniji** (prljavstina i rukavice nisu problem), ali se mora bas pritisnuti. Drugim recima dobri su u slucajevima kada su ruke prljave. Mozemo pritiskati cime god. Stalno menjamo otpor struje, stalno vucemo nesto iz baterije.
Kapacitivni su napravljeni kao kondenzator izmedju 2 providne folije. Elektrode folija su providne i imamo elektrode konderzatora. Svaki kondenzator je velicine polja najmanje reakcije ekrana. Kada prislonimo prst na kapacitet kondenzatora se dodaje nas kapacitet. Redno vezani kapaciteti se sabiraju kao ona suma reciprosnih vrednosti (paralelna veza otpornika) i ukupan kapacitet bi bio manji od najmanjeg otpornika u vezi. Lako se detektuje i ne trosi dodatnu struju, a ne moramo nista jako da stiskamo. Manje su otporni, moramo pritiskati provodnikom (npr. sa rukavicama ne radi). Uticu prljave ruke. Zahteva posebne styluse. Ako korisniku dajemo upuststvo da koristi trebamo mu reci da **dodirne** to sto je potrebno.

***Multi-touch** :
Bilo je nemoguce napraviti veoma velike kapacitivne ekrane. Zato su uzeli kvalitetnu plasticnu foliju, Mora sve biti obradjeno savrseno, bez neravnina (kao ogledalo) i onda se stave infracrvene (zato sto ga ne vidimo) diode. Kada

stavimo na povrsinu prst gustina se menja i vise infracrvene svetlosti ide ka njemu. Na osnovu promene tog infracrvenog svetla kamera koja se nalazi u uredjaju detektuje gde smo stavili prst. Mogu biti ogromni ekrani, cak i velicine celog zida.

Uredjaji indirektne kontrole:

***Mis** :
U kutiji zatvorene dve okrugle ploce, pomeranjem su se ti tockovi vrteli, pomerale su se osovine, na krajevima osovina se nalazio krug koji je bio izbusen. Sa jedne strane njega je bio izvor svetlosti, sa druge strane foto-celija i prekidanjem tog svetlosnog snopa slali su se impulsi i ocitavanjem tih impulsa se moglo ustanoviti pomeranje po obe koordinate i pretvaralo u pokret.
Glavni nedostatak je bio sto su limeni diskovi grebali radnu povrsinu. Njih je zamenila gumena loptica koja se cesto prljala. Takav mis je zamenjen laserski (bacao je lasersku lepezu) I odbijanjem od povrsine se detektovalo u koju stranu se krece. Problem kod laserskog je bio sto podloga nije smela biti iste poje kao laserski snop (najcesce crveni). Danas se koristi opticki. On bukvalno ima mali video cip koji snima povrsinu, poredjenjem frejmova dobija vektor pokreta.
Misevi mogu biti zicni i bezicni. Prednost bezicnog je sto nismo ograniceni kablom, ali je problem sto radi na baterije koje se mogu isprazniti bas kada nam je najpotrebniji. Treba mu velika povrsina za upotrebu, da bismo ga vrteli i pomerali.

***Trackball**:
“Lopta za pracenje”, verzija misa kojeg su okrenuli na ledja tako da mu se vidi samo loptica velicine skoro kao teniska loptica da bismo ga pomerali prstom tako da moze detektovati I jako male i precizne pokrete. Moze i da pravi velike pokrete. Moze biti integrisan u laptop i/ili tastaturu. Glavna mana je sto zahteva neznost i ciste ruke.

***Joystick**:
Nastao pre misa (preuzet iz vojske u sistemina za upravljanje raketama), preuzet iz vojske, bila je trka izmedju njega i misa ko ce postati standardni pointerski uredjaj. Ima najmanju povrsinu za upotrebu, moze se drzati u ruci. Postoje 2D (upravljanjem palicom mi upravljamo pointerom, a moze imati i specijalizovane dugmice) I 3D joystici (izvlacimo ga i guramo dole i tako pomeramo po trecoj dimenziji). Moze biti integrisan u uredjaj za igranje (gamepad). U njih moze biti ugradjen I neki vibracioni uredjaj u cilju davanja boljeg feedback-a.

***Trackpoint** :
Ima bas najmanju povrsinu. Izometrican joystick u tastaturi. Prakticno se ne uopste pomera, vec se deformacijom (crvene) provodne gume i to je input do naseg pokreta. Moze biti i taktilan, tako sto se u tu gumu ugrade iglice i izbocavanjem tih iglica (programski smo mogli da uticemo na njih da se izbace i spustaju) korisnik oseti kakve je povrsine ekran preko koje prelazi kursor koji pokrecemo, tj. gde se na ekranu trenutno nalazi kursor (zgodan ljudima sa ostecenim vidom jer mogu da 'opaju' svoj ekran). Taktilan je i veoma skup. Podlozan kvarovima jer ima jako puno mehanickih delova.

***Graficke table**:
Povrsina po kojoj mozelo da pomeramo neki uredjaj, u zavisnosti od tog uredjaja moze biti elektronska (sa galvanskim spojem izmedju citave povrsine table po kojoj se krecemo i pointerskog uredjaja (olovke) i tako se prati pokret koji se prave), kapacitivna (ne sa galvanskim spojem zbog promene kapaciteta) ili elektro-akusticna (salju se i primaju zvucni signali sa kraja na kraj table, na krajevima se nalaze zvucnici i usmereni (jer salju zvuk u jednoj liniji) i mikrofoni (primaju od svog zvucnika zvuk), olovcicom mi prekidamo (apsorbujemo) te signale). Mozemo crtati pomocu njih jer obezvedjuju dobar pokret.

***Touchpad**:
Integrisani su u uredjaje. Kapacitivan je sistem, prelaskom prsta po povrsini menjamo kapacitet (delimo kapacitet sa prstom) i tako se detektuje pokret. Toshiba napravila touchpad koji je davao neke graficke prikaze, poput kalkulatora ili omogucuje upravljanje CD-drajvom (audio-player). Zato, kada bi pao operativni sistem, ove dve funkcije bismo mogli koristiti bez potrebe za podizanjem novog OS.

Fitts-ov zakon:

Industrijski psiholog, bavio se pitanjima optimizacije rada, tj. mentalnim i fizickim opterecenjem coveka dok radi. Napravio je studiju od organizacije radnog mesta neke fabrike za proizvodnju finih mehanickih uredjaja. Studija se ticala organizacije radnog mesta coveka da bude sto manje opterecen i da proizvodi sto manje skarta.
Pravio je nekoliko serija merenja nad jednim jednostavnim zadatkom. Na jednoj plozi od izolatora stavio je dve provodne ploce koje su imala krilca kako bi se mogle suzavati i razmicati. Imali smo i jednu olovku, takvu da su olovka i provodna ploca predstavljale prekidac. Korisnik je imao zadatak da sto brze moguce naizmenicno dodiruje oba provodna polja. Polja su se razmicala i suzavala i merene su akcije korisnika. Na osnovu toga, izveo je jednacinu vremena pointiranja. Vreme je srazmerno distanci (sto je distanca izmedju cilja i odatke smo poceli veca to je vreme duze), obrnuto srazmerno sirini targeta koji gadjamo(ne moramo da "kocimo" prilikom gadjanja, pa lakse pogodimo cilj) -> Parametri objasnjeni na slajdu. Zasto sto je siri target koji gadjamo, je vreme manje? Jer mi pokrenemo uredjaj i brze ga pogodimo, opusteniji smo jer ne moramo previse da kocimo.
Danas Fitts-ov zakon sluzi da proizvdjaci **pointerskih uredjaja** (I mobilnih telefona) na ekranu definisu I rasporede polja I mere akciju korisnika. Na osnovu toga, mogu menjati dizajn uredjaja tako da se postigne sto manje vreme pointiranja. Koristi se I prilikom organizacije korisnickog **Menu-a**(padajuci ili "pie" menu -> sve opcije su na istoj razdaljini pa je vreme pointiranja prema svima isto i samim tim vreme proracuna je isto za sve).
Primenom Fitts-ovog zakona mis je postao standardni pointerski uredjaj. Ono sto je presudilo je sto je covek pravio manji broj gresaka.

Copy uredjaji

Podela po tome kakav dokument salju korisniku.

***E-copy (elektronski)**: Monitori, Beam projektor, Skener…

1. **CRT monitori** :
   Prvi nastali, snop elektrona krece se sa katode na anodu, skrece i udara u ekran sa fosfornim slojem. Spolja se nalazi maska(na pozitivnom naponu) koja sluzi za antilansing, kako ne bi doslo do rasipanja oko piksela. Osnovna tehnologija je fosforna, proizvodi finu svetlost velikog kontrasta prikaza, ugao gledanja je veliki. Ekran nije potpuno ravan, vec blago zakrivljen.I dalje je medju najbrzima. Mane su jer zauzima mnogo prostora. Sto je ekran veci mora biti deblji monitor (veci gabariti), elektronski snop treba da predje veci put kako bi skrenuo do svih ivica. Fosforna tehnologija se mora osvezavati i spada medju najvece potrosace. *Ako ocemo da uticemo na osvetljenje, tj. kolicinu svetlosti koja izlazi sa ekrana mi moramo da smanjimo broj elektrona, tj. manje elektrona ce pobuditi neku tacku fosfora i manje ce preneti energije i fosfor ce slabije svetleti.

2. **Plazma ekrani**:
   Zameninili CRT, najcesce televizori. Zasniva se opet na fosfornoj tehnologiji, ali su ga zatvorili u male camcice (posudice). Jedan piksel je predstavljen sa 3 posudice jedna do druge, na cijem dnu se nalazio fosfor sa odredjenim primesama kako bi se mogli emitovati plavi, zeleni I crveni fotoni. Fosfor se sada pokrece (pobudjuje) elektrodama koje se podizu na veoma visok napon. Joni u plemenitom gasu(plazmi) pocnu u odredjenom momentu da provode struju, napon blago opadne. Neophodan je visok napon da se upali, ali se onda jako lako odrzava. Namaze se fluroscentnim namazom iznutra da svetlelo belo. Jako je vazno taj gas dooobro zatvoriti u te camcice da ne curi, jer ako padne ispod neke kriticne vrednosti dobicemo mrtvu celiju, jer ce onda iscureti gas i ne ce ga biti dovoljno da se pobudi.
   Dobro je bilo sto se to sve spakuje na svega nekoliko centimetara. Debljina celog monitora je vise zavisila od mehanike koja je bila potrebna za drzanje I zastitu velikih ekrana, ne vise od same prikazne tehnologije.
   Boje I dalje fine (jer fosfor nosi boju), kontrast fin, ugao gledanja nije tako dobar, potrosnja manja od CRT-a.
   Problem je bila brzina (pogotovo paljenja).
   Problem su bile "Fantom slicice" koje su se javljale kada se fosfor potrosi. Primer fantom slicica je Univerzijada u

Beogradu. Kada se zavrsila, prodavali su ogromne plazma ekrane na kojima su se prikazivali rezultati utakmica. Posto su se za prikaz rezultata koristili gotovo uvek iste celije I kada bi neko upalio normalan program, tacno bi se videle koje su celije najvise potrosene. Dale bi bledje boje. Drugi primer je bio stalni prikaz televizijskog programa sa rezolucijom takvom da neki deo ekrana stalno ima "crni ram". Onda kada bismo npr. pustili DVD film koji koristi ceo ekran, leva I desna povrsina ekrana bi lepse sijale, jer fosfor nije potrosen. (4x3 vs 16x9 dimenzije)
Isto, javljali bi se mrtvi pikseli. Postojala je garancija do 3000h gledanja. Posle toga krenu da crkavaju kao zaraza. Dovojlno je da mala pukotina nastane negde i ona ce da se siri dalje. Moze da nastane pukotina na camcicima ili na vezi izmedju camcica i provodnih pokrivaca.

3. **LCD**:
   Liquid Cristal Display. Prikazni uredjaj sa tecnim kristalima. Kristalne strukture imaju obicaj da sprovode svetlo. Kako bakar sprovodi struju, tako kristalna resetka sprovodi I prelama svetlo. Kristal ovde nije u pravom smislu reci tecan, vec mekan pa su veze izmedju kristalnih resetaka slabe, tj. cvorovi kristalne resetke se mogu zavrtati u elektricnom polju.
   LCD celija propusta ili ne propusta svetlost. Boju svetlosti diktira drugi izvor.
   Kristalna resetka se sece I postavlja izmedju dva polarizatora sa elektrodama, tako da moze da se rotira tako da moze da predje iz svog vertikalnog polozaja u horizontalu. Sa jedne strane se nalaze horizontalni, a sa druge vertikalni polarizatori. Kada nema napona na resetkama, stoji u svom "isecenom" stanju. Svetlost koja dolazi na resetku ima svoju horizontalnu I vertikalnu komponentu. Kroz vertikalni polarizator prolazi samo njena vertikalna komponenta, krenuce da prenosi kroz ostale kristale i prolazeci kroz kristale ce se okretati zajedno sa njima (potpuno u horizontalu). Na drugom kraju se nalazi horizontalni polarizator koji propusta samo horizontalnu komponentu svetlosti (koje ima dosta). Napolje izlazi ta svetlost. Ovo je nema struje!
   Kada dovedemo staticko elektricno polje na resetku, cvorovi ce zauzeti takav polozaj da ga elektricno polje najmanje dodiruje. Polje ne mora biti narocito jako. Igranjem sa ovim naponom mi odredjujemo koliko cemo svetlosti propustati I koliko ce svetlosti prolaziti kroz LCD kristal. Starijim varijantama je trebalo vremena da se pokrenu, tj da isrotiraju I postave sve kako treba.
   Imamo jedan izvor svetlosti koji mora biti beo (neonska lampa, u pozadini televizora imali smo difuzor koji je razbacivao svetlost od neonke tako da ona prolazi kroz svaku jedinicu povrsine prikaza). Unutar svake celije nalaze se ogledala koja su filter plave i crvene komponente (npr. crveni filter propusta crvenu a reflektuje plavu i zelenu, dok plavi filter propusta plavu, itd.). Imamo 3 LCD celije, gde jedna dobije plavu, jedna zelenu i jedna crvenu. Unutra je prizma koja sabira komponente i sumu sve tri salje na prikaz.
   Kasnije se ovaj mehanizam unapredio I dobili smo LED lcd ekrane. Koriste P-N spoj(na p komponenti postoji visak elektrona, a na n komponenti manjak, kada se uspostavi napon izmedju dve, materijal svetli). Diode svetle belom svetloscu I postavljene su po ivicama ekrana (edgelit) ili su postavljenje po citavoj pozadini ekrana (backlit). 1 dioda obicno pokria 8px.
   Sto se tice ustede, kod standardnih sa lampom ta lampa se nikada nije gasila, EdgeLit bi gasio samo kada je ceo red crn, a kod BackLit-a mogu da se gase za svaki crni piksel. Ali generalno im je potrosnja struje manja. Problem je sa belom svetloscu, pogotovo kod standardnih, jer se jako greju od svetla, tj. sunca.
   Sve je brzi i brzi (ali jos nije kao CRT brz).

   *Pitanje : Imamo sve ove pobrojane ekrane. Pratimo potrosnju. Gledamo film. Dodje neka scena na kojoj padne mrak I traje par sekundi. Kod kojih potrosnja pada kod kojih ostaje ista?
   *Odgovor: CRT opada, ali jako malo. LED LCD ce skroz opasti, dok ce se standardnom LCD povecati jer se svi kristali moraju prebaciti u polozaj u kom ne provode (on kad provodi najvise trosi).
   *TFT: tranzisotrska logika je jako brza

4. **OLED** :
   Ubacivanje organske (klasican LED je neorganska) hemije u P-N spoj I otkrili da naponom mogu upravljati koji ce

fotoni da svetle dok struja prolazi kroz diodu. Pred probijanje P-N spoja sija bela svetlost. Doslo se cak do 1 diodica 1 px. Koriste se na manjim povrsinama poput mobilnih telefona. Osvetljenost dobra, mala potrosnja, ugao posmatranja dobar, mali gabariti.

5. **E-ink**:
   Svuda gde nam ne treba boja. Izmedju 2 providne povrsine (gornji da zastiti uredjaj spolja, a donji da da cvrstinu da stoji) nalaze se loptice unutar kojih je u jos manjim lopticama mastilo u 2 boje (najcese bele I crne). Jedna loptica – 1 piksel.Obe boje loptica su suprotno naelektrisane. Na donjoj povrsini menjamo napon I tako kontrolisemo boju mastila koje ce se popeti na povrsinu. Ako npr. imamo crne I bele loptice gde su crne pozitivno, a bele negativno naelektrisane dovodjenjem pozitivnog napona privucice se bele na dno, a crne oterati ka gornjoj povrsini I taj piksel bismo videli kao crni.
   Moguce je npr umesto 1 elektrode ispod jednog piksela smestiti 4 manje elektrode I tako kontrolisati nivoe sive. Potrosnja nikakva, ali ovde nam je neophodan spoljasnji izvor svetlost da bismo videli sta je na ekranu. Zasniva se na refleksiji svetlosti iz okoline. Ne mogu se koristiti u mraku.
   Kasnije nismo vise imali 2 vrste loptica, vec samo jednu vrstu (belu)koje su uronjene u mastilo. Radi na istom principu kao do sad.
   Isto, napravljena je struktura takva da nam pikseli sada imaju 2 nivoa. Gornji, u koji mi gledamo, tu dolazi svetlost koja se reflektuje. Izmedju 2 nivoa se nalazi ogledalo. Ispod ogledala se nalazi mastilo. Naponom kontolisemo koliko mastila ce iz donje komore preci (sto je napon veci) preko ogledala I kada se svetlost reflektuje vidimo odredjeni nivo sive (moze i druge boje u zavisnosti od mastila).

6. **Holografski prikazi**:
   Prikaz u 3D prostoru. Imamo izvor ravnih talasa I tackasti izvor svetlosti. U svetlosti se nalaze podaci, tj. ono sto treba prikazati/memoriseti/preneti. Na mestu uktstanja ovih talasa oni interaguju I formira se stojeci talas odredjenih vrednosti. Vrednost je diktirana tackastim izvorom, a ravan talas diktira prostor u kom ce se oni sabirati. Ako se neki fotoosetljivi matrijal stavi u taj prostor, ako dovoljno dugo stoji u tom prostoru njegova resetka ce se trajno deformisati kako je definisano u onim podacima koji se prenose. Na mestu reprodukcije se na taj kristal pod istim uglom treba dovesti isti tip ravnih talasa kakav je bio pri upisu, a on ce prelamati te ravne talase po supljinama na zapisu I nastace tackasti talas istih karakteristika kakav je bio upisan.
   To su holografske memorije. Ne dobijamo niz nekih vrednosti, vec citavu ravan podataka.

   *SLM : Prostorni modulator svetlosti kojim upravlja racunar. On izvor svetlosti koji dolazi u prostoru modulira tako da dobijemo trecu dimenziju, tj. oscilacije svetlosti mi vidimo kao 3D sliku.

   *Apple : Prikazne tehnologije izmedju 2 sfericna ogledala. Prikaz se propusta kroz lupu, pa se na skromnoj povrsini(ekran mobilnog telefona) moze generisati dobar prikaz. Postoje I senzori pokreta, pa se moze obavljati I interakcija prstom.

   * Problem je sto se sve to ne vidi tako kvalitetno u realnom okruzenju i samim tim ne moze najbolje da se interaguje s tim. Problem je fizika, tj. optika koja zahteva razdaljinu. Cilj je da se to optimizuje da bude primenjivo u danasnjim racunarskim tehnikama.

   *Postoje razlicite definicije rezolucije. Najcesca je oba sto smo imali, broj piksela po visini I po sirini, moze se uvesti I dijagonala. Sve se izrazava u incima.
   *PPI : Gustina piksela na prikazu, tj. rezolucija po jedinici povrsine (pixel per inch).
   *Dot per Inch : Razdaljina izmedju istih komponenti trijade susednih piksela. Odredjuje razdaljinu za konzumiranje ekrana (za moblinki je mali, dok je za velike ekrane veci). Sto je dot per inch manji pikseli su manji (za mobilni tj. manje razdaljine) i obrnuto.

7. **Fotoaparati/kamere** :
   Imaju optiku kojom zumiramo I podesavamo opticku razdaljinu tako da najvise detalja uzima sa neke razdaljine od sebe. Svetlost koja se reflektuje sa uredjaja pada na video cip (CCD ili CMOS) koji onda konvertuje svetlost u elektricni signal. CCD su veci, deblji, ma im rezolucija ne moze biti velika, ali prirodnije konvertuje svetlost I ima lepsi prikaz. CMOS se sastoje iz tranzistora I 2 kondenzatora, moze biti manje povrsine, veca rezolucija, ali naglasavaju svaku promenu u osvetljaju "onoga u sta gledaju".
   Primer : Slikamo/snimamo put I prolaze kola sa ukljucenim farovima. Ako imamo CCD videcemo normalan odbljesak. Malo ce nas zalsepeti pa proci dalje. Dok ako imamo CMOS(verovatnije je) on ce taj odbljesak prikazivati duze I intenzivnije nego sto bi realno bio. Moze se softverski ili hardverski malo dopraviti.

   *HDR slike : High Dynamic Range. Slika je velikog opsega dinamike, tj. dobro hvata I tamne I svetle delove ambijenta. Radi to tako sto ce napravit nekoliko brzih uzorkovanja realnosti sa razlicitim otvorima blende. Onda se pomocu nekog algoritma se te slike spajaju. Nikada nece imati takve detalja kao da smo odredjeni deo fokusirali, jer se spajanjem slika detalji gube.

8. **Skeneri** :
   Pretvaraju H-copy u E-copy. Mogu biti refleksioni skeneri i transparentni.
   Refleksioni : Skeniranje papira. Stavimo dokument koji treba da se skenira I preko njega ide neonka. Ona obasjava dokument, od njega se reflektuje svetlost ka njegovom sadrzaju. Tu svetlost primaju CCD ili CMOS davaci koji ce primiti red svetlosti I pretvoriti ga u red piksela.
   Transparentni : Skeniranje rendgenskih snimaka. Primalac i davalac svetlosti su sa suprotne strane dokumenta. Dokument mora biti transpatentan. Svetlost prolazi kroz dokument I ide na primaoca i pretvara u elekronski zapis.

9. **Beam Projektori** :
   *LCD tipa : Velika jaka lampa prolazi kroz LCD ekran. On malo ugusi tu svetlost. Nikada ne mogu biti tako dobri da se mogu projektovati u lose osvetljenoj prostoriji I potrebno je vreme da se LCD pripokretanju okrene kako treba za projekciju.
   *DLP projektori : Zasnivaju se na MEMS cipovima I sa gornje strane ima mala ogledala. Jedno ogledalse jedan piksel. Sva ogledalca su usmerena u istu stranu inace, ali se programski mogu naterati pojedinacna ogledalca da se okrenu u drugu stranu. Imaju svoj mirni I radni polozaj. Svetlost se iz nekog izvora usmerava na ta ogledalca. Ona koja treba da svetle treba da budu u mirnom polozaju. Sa ogledalaca koja su u radnom polozaju se reflektuje svetlost na metalnu crnu plocu (crna je da bi privukla svetlost, metalna da dobro priovodi toplotu I da se hladi), pa svetlost prolazi kroz sociva I osvetljava tackicu na zidu. Ako treba boja radice se difrakcija (valjda se ovako kaze!) svetlosti, pa kroz okruglu lepezu koja se vrti velikom brzinom i tako menjati osvetljenost u zavisnosti boje. Brzi su i manji od LCD.

10. **HMD Virtual Reality** :
    Montira se u visini ociju, covek u to gleda, prikaz se moze emitovati na jedan veci vizir ili za svako oko. Sa vizirom je bolji dozivljaj jer nemamo svi istu sirinu i velicinu ociju. Moze se napraviti cak i od mobilnog telefona.

11. **HMD Augmented Reality :**
    Kamera mobilnog moze snimati okolinu i napraviti neki vid augmented reality dodavajuci na snimljeno neke virtuelne objekte. Mogu se koristiti i providne naocare gde se na staklo tih naocara se projektuje graficki prikaz. Vidimo realno, a taj graficki prikaz je virtuelna komponenta. To je praceno kamerama i zvukom da moze biti interaktivno.

*Hard-copy : na papiru, "mozemo staviti prst na njih"

1. **Impact-line stampaci :** Radili kao pisace masine. Jedan od najbrzih. Veliki, ogroman cilindar gde su u redovima rasporedjeni isti znaci (npr. ceo red malih slova “a”). Cilindar moze da se okrece velikom brzinom. Imamo mastiljavu traku, papir I red cekica sa suprotne strane. Dok se cilinda vrti, red cekica udara sve pozicije na kojima bi se nalazilo slovo (ili broj, znak) koje se nalazi u redu slova na cilindru koje je u datom trenutku okrenuto ka papiru.Primer : Okrenuto je slovo “a” u datom trenutku I cekici ce dok se cilindar ne pomeri na naredno slovo udariti sva mesta na kojem bi se javilo slovo “a”.Kada cilindar napravi jedan krug, taj red je sigurno odstampan. Papir se pomeri I stampa se sledeci red. Velicina papira je unapred odredjena.
   Problem je ogromna buka lupanja cekica I nije mogao da stampa graficke prikaze (samo alfanumerike) i boja stame je bila boja mastiljave trake. Bio je veoma velik.

2. **Dot-matrix:**
   Imali smo kolone iglica(9 -> 1 kolona, 12 ->2 kolone ili 24->3 kolone) koje su lupale po mastiljavoj traci. Oblik koji ce se odstampati zavisi od toga kako smo isprogramirali. Mogu da stampaju grafiku, boja zavisi od mastiljave trake. Manje bucan i manjeg gabarita od prethodnog, ali I sporiji. Danas se jos koristi u posti/banci/menjacnici za overu uplatnica(ostavlja trag na svim verzijama u istom trenutku) i za stampu fiskalnih racuna.

3. **Termalni stampaci:**
   Nema sada udarca, vec umesto iglica imamo male brze otpornuke koji se greju I hlade u zavisnosti od napona koji se na njih dovedu. Oni isprze papir. Crno-bela stampa. Papir moze biti I premazan nekom hemijom, onda od duzine przenja menja se I boja (desetine razlicitih). Moze da stampa grafiku. Jako je tih. Malo brzi je od dot-matrix, ali koristi specifican papir (ako ocemu u boji skup je) koji moze da se “przi” i nikako ne mozemo istovremeno stampati 2+ primeraka (nema kopije, kao kod uplatnice). Isto, prikaz nije trajan. Moze da izbledi ili potpuno pocrni (fiskalni racun).

4. **Inject stampaci:**
   “Pljuckaju” boju. Postali vrlo jeftini. Ali zahtevaju kupovinu i/ili pinjenje kertridza i papira za stampanje. Mogu stampati bilo sta, cak I fotografiju jer pljuckaju boju na papir. Dugotrajni su, ne blede kao raniji termalni stampaci. Moguce je obloziti ih topljivom plastikom u cilju zastite stampe.
   Mogu biti termalni ili pizioelektricno kreirati kapljice.
   *<u>Termalni</u> : Predstavnik Canon. Svaka boja je imala svoju “pipicu” iz koje treba da izadje kap boje. Bola ce izlaziti kada se ta pipica zagreje.
   *<u>Piezoelektricno</u> kreiranje : Predstavnik je Epson. Sa obe strane pipice se nalaze elektrode. Kada nema elektricnog polja izmedju, nema ni pljuckanja mastila. Kada se polje uspostavi, dolazi do piezoelektricnog efekta, mastilo ne voli da ostane u polju, pobegne i kapne na papir.
   Obe vrste mastila imaju veoma razlicite karakteristike, zato se ne sme sipati u kertridze drugog proizvodjaca.

   *Pitanje : Rezolucija stampe je uslovljena kvalitetom papira, zasto? Ovi stampaci su zasnovani na pljuckanju boje na papir. Ako nije glatka povrsina, kapi boje nece lepo I ravnomerno padati jedna preko druge, vec ce se razlivati. Ako nam je bitan kvalitet, treba se koristiti foto-papir (jako upija).

5. **Laserski stampaci:**
   Ostavljaju otisak koji je na negativ napravio laser. U sredini se nalazi fotoosetljivi bubanj. Na njega pada laserski zrak, red po red osvetljava bubanj sa sadrzajem (na onim mestima na kojim treba da se napravi stampa). Delovi bubnja na koje je pao laserski zrak se malo naelektrisu. Pored se nalazi toner u kom se nalazi fini prah(osusena stamparska boja) koji izlazi kroz mali otvor. Bubanj kako se okrece, naelektrisanim delovima kupi malo tog praha. Papir prolazi direktno ispod bubnja I osovina koja vuce papir ujedno ga I greje. Zagrejan papir dodiruje bubanj, istopi taj prah na sebe I izlazi napolje. Transportnom trakom se skida se visak farbe sa bubnja I razelektrise se kako bi bio spreman da se naelektrise novim sadrzajem.
   Ovako radi crno-beli, ako jeu boji onda ima vise tonera koji nanose prah. Transportnu traku je potrebno redovno

menjati da se ne bi stalno javljali ostaci od prethodnog stampanja tonera na fotoosetljivom cilindru, pa ce se on svaki put dodatno isprziti (zagrejati) i praviti vecu stetu. Ima ogranicen vek trajanja.

6. **3D stampaci:**
   Stampaju u volumenu ili bace prašak na neku povrsinu, po toj povrsini prelazi laser I topi prasak(nakon topljenja on ocvrsne), duvalica oduva visak praska I nanosi sledeci sloj (nivo praska). Umesto praska moze se koristiti I zica od plastike koja se topi I tako gradi 3D objekat koji treba. Poceli su da rade I sa organskim materijalima, tako da ce se u buducnosti mozda moci izgraditi neki organ ili deo tela.

7. **Plotters:**
   Prava su “slika” ljudi. Imamo most I rucicu koja se krece preko papira I iscrtava sadrzaj. Moze stampati I na ravnim povrsinama, zakrivljenim(cilindricnim) ili na “beskonacnoj” traci (pomeranjem rolne koja moze biti veeeoma dugacka). Za inzenjersku grafiku odlicni.

Uredjaji za korsnike sa posebnim potrebama

Prilagodjavanje uredjaja korisnicima koji imaju ogranicenu mobilnost. Cesto to malo pokreta koji oni mogu da naprave su vrlo precizni. Sa druge strane, mozemo imati I korisnike koji imaju veci pokret, ali su nekontrolisani (jako udaraju, tesko ciljaju…). Prave se zato tastature sa velikim tasterima od izdrzljivog materijala. Prave se specijalizovani uredjaji za one koji nisu u mogucnosti da koriste ruke ili noge, poput joysticka koji se moze pomerati jezikom a klik se odvija udisanjem I izdisanjem. Prilagodjavanje ekrana za ljude sa ostecenim vidom, sluhom itd…, npr. digitalne table na koje se stave folije koje su digitalizovano zapisane koje interpretiraju sta je gde pritisnuto.