



Odsijecanje i scan konverzija

Dr. sci. Emir Skejić, vanr. prof.
Fakultet elektrotehnike Tuzla



Rendering trougla

- Glavne faze u *tradicionalnom grafičkom cjevovodu* su:
 - Transformacija
 - Osvjetljavanje
 - Odsijecanje / Culling
 - Scan konverzija
 - Rendering piksela



Transformacija

- U transformacijskoj fazi, vrhovi se transformiraju iz njihovog originalnog prostora objekta preko niza koraka u finalni "2.5D" prostor uređaja od stvarnih piksela

$$\mathbf{v}'_{(4D)} = \mathbf{P} \cdot \mathbf{C}^{-1} \cdot \mathbf{W} \cdot \mathbf{v}$$

$$\mathbf{v}'' = \begin{bmatrix} \frac{v'_x}{v'_w} & \frac{v'_y}{v'_w} & \frac{v'_z}{v'_w} \end{bmatrix}$$

$$\mathbf{v}''' = \mathbf{D} \cdot \mathbf{v}''$$



Transformacija: Korak 1

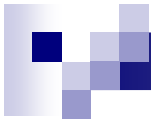
$$\mathbf{v}'_{(4D)} = \mathbf{P} \cdot \mathbf{C}^{-1} \cdot \mathbf{W} \cdot \mathbf{v}$$

- \mathbf{v} : Originalni vrh u prostoru objekta
- \mathbf{W} : Matrica koja transformira objekt u svjetski prostor
- \mathbf{C} : Matrica koja transformira kameru u svjetski prostor (\mathbf{C}^{-1} će transformirati iz svjetskog prostora u prostor kamere)
- \mathbf{P} : Neafina matrica perspektivne projekcije
- \mathbf{v}' : Transformirani vrh u 4D nenormaliziranom prostoru pogleda
- Napomena: Ponekad ovaj korak biva podijeljen u dva (ili više) koraka. To se često radi da bi se proračuni osvjetljenja i odsijecanja obavili u prostoru kamere (prije primjene neafine transformacije)

Transformacija: Korak 2

$$\mathbf{v}'' = \begin{bmatrix} \frac{v'_x}{v'_w} & \frac{v'_y}{v'_w} & \frac{v'_z}{v'_w} \end{bmatrix}$$

- U narednom koraku preslikavaju se tačke iz 4D prostora u normalizirani prostor pogleda, pod imenom *prostor slike*, koji se kreće u intervalu od -1 do 1 po x, y i z
- Od sada pa nadalje, o tački ćemo uglavnom razmišljati kao da je 2D (x i y), s dodatnim informacijama o dubini (z). Ovo se ponekad naziva 2.5D



Transformacija: Korak 3

$$\mathbf{v}''' = \mathbf{D} \cdot \mathbf{v}''$$

- U završnom koraku transformacijske faze, vrhovi se transformiraju iz normaliziranog $-1 \dots 1$ prostora slike i preslikavaju u stvarni pravougaoni *viewport* od piksela



Transformacija

$$\mathbf{v}'_{(4D)} = \mathbf{P} \cdot \mathbf{C}^{-1} \cdot \mathbf{W} \cdot \mathbf{v}$$

$$\mathbf{v}'' = \begin{bmatrix} \frac{v'_x}{v'_w} & \frac{v'_y}{v'_w} & \frac{v'_z}{v'_w} \end{bmatrix}$$

$$\mathbf{v}''' = \mathbf{D} \cdot \mathbf{v}''$$



Odsijecanje i culling



Odsijecanje

- Neki trouglovi će u potpunosti biti vidljivi na ekranu, a neki mogu biti potpuno izvan pogleda
- Neki trouglovi mogu presijecati rub ekrana te zahtijevaju specijalno rukovanje
- Prostor vidljiv kamerom obrazuje volumen koji se naziva *volumen pogleda*. Trouglovi koji sijeku granice volumena pogleda moraju biti odsječeni.
- Povezani proces *selektivnog odbacivanja* (engl. *culling*) odnosi se na određivanje koji primitivi su potpuno nevidljivi
- Izlaz procesa odsijecanja/cullinga je skup vidljivih trouglova koji se nalaze unutar dimenzija prikaznog uređaja

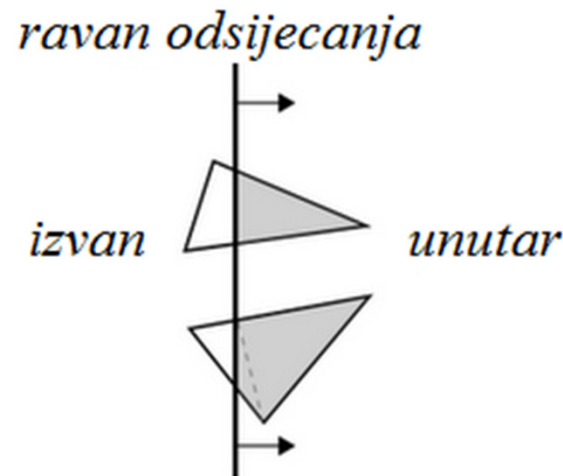


Odsijecanje

- Trouglovi se općenito odsijecaju jedan po jedan, mada složenije implementacije trouglove mogu odsijecati u grupama
- Mi ćemo razmotriti samo odsijecanje jednog trougla
- I perspektivni i ortografski volumeni pogleda su omeđeni sa 6 ravni, te se mogu tretirati na vrlo sličan način
- Jedan trougao bi potencijalno mogao probadati svih 6 ravni (mada bi u praksi bilo prilično neuobičajeno da trougao probada više od 2 ravni)
- Ustvari, broj ravni i nije toliko bitan, budući da osnovni algoritam jedanput odsijeca na svaku ravan

Odsijecanje

- Ukoliko trougao probada određenu ravan odsijecanja, nastat će ili jedan ili dva nova trougla
- Dobijeni trouglovi zatim se testiraju na preostale ravni odsijecanja





Odsijecanje

- Da bi trougao odsjekli na određenu ravan odsijecanja, prvo moramo odrediti na kojoj je strani ravni svaki od njegova 3 vrha
- Ako su sva 3 vrha "unutar", tada trougao ne mora biti odsječen
- Ako su sva 3 vrha "izvan", tada je trougao potpuno nevidljiv i biće odbačen
- Ako je samo 1 vrh unutar, tada moraju biti (privremeno) kreirana dva nova vrha pa će biti kreiran i samo jedan novi trougao
- Ako su 2 vrha unutar, tada moraju biti kreirana dva nova vrha kao i dva nova trougla
- Oba slučaja odsijecanja uključuju određivanje gdje 2 stranice trougla sijeku ravan, te se tako mogu i tretirati na vrlo sličan način



Odsijecanje: Testiranje unutar/izvan

- Ravan odsijecanja može biti definirana pomoću tačke \mathbf{p} na ravni i jedinične normale \mathbf{n} (mi ćemo odabrati \mathbf{n} tako da pokazuje prema unutrašnjosti volumena pogleda)
- Da bismo testirali da li su vrhovi trougla unutar ili izvan ravni odsijecanja, računamo označenu udaljenost do ravni za svaki od 3 vrha \mathbf{v}_0 , \mathbf{v}_1 i \mathbf{v}_2

$$d_0 = (\mathbf{v}_0 - \mathbf{p}) \cdot \mathbf{n}$$

$$d_1 = (\mathbf{v}_1 - \mathbf{p}) \cdot \mathbf{n}$$

$$d_2 = (\mathbf{v}_2 - \mathbf{p}) \cdot \mathbf{n}$$

- Pozitivna (ili 0) udaljenost indicira da je vrh unutar volumena pogleda, a negativna udaljenost indicira da je izvan



Odsijecanje: Presijecanje stranica/ravan

- Da bi se odredila tačka \mathbf{x} u kojoj stranica trougla probada ravan, potrebna su dva vrha \mathbf{v}_a i \mathbf{v}_b koja obrazuju stranicu, kao i njihove označene udaljenosti d_a i d_b

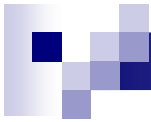
$$t = \frac{d_b}{d_b - d_a}$$

$$\mathbf{x} = t\mathbf{v}_a + (1 - t)\mathbf{v}_b$$



Odsijecanje: Prostori

- Odsijecanje može biti urađeno u bilo kojem željenom prostoru
- Uobičajeno je odsijecanje obaviti u prostoru kamere, budući da je to regularan 3D prostor u kojem će ravni odsijecanja biti zgodno opisane
- To zahtijeva transformiranje vrhova u prostor kamere, potom odsijecanje, a zatim transformiranje vrhova u 4D nenormalizirani prostor pogleda
- Ukoliko je pojedinac domišljat, postupak se može ubrzati i odsijecanje obaviti u 4D nenormaliziranom prostoru, što eliminira potrebu za transformiranjem vrhova preko dvije zasebne matrice



Odsijecanje: Setup

- U pripremi za proces odsijecanja korisno je unaprijed izračunati normale za 6 ravni u volumenu pogleda
- To se može uraditi jedanput po frame-u kada je matrica projekcije specificirana (prije nego što počne stvarni rendering)



Culling

- U računarskoj grafici, *selektivno odbacivanje* (engl. *culling*) se odnosi na postupak određivanja onoga što *nije* vidljivo
- Čim prije se ustanovi da trougao neće biti vidljiv, manje će se vremena potrošiti na njegovu obradu
- Culling se obavlja na nivou pojedinačnih trouglova, međutim, može se obavljati i na nivou objekta
- Ako možemo brzo odrediti da li cijeli objekt leži izvan ekrana, tada uopšte ne moramo procesirati bilo koji vrh ili trougao
- Za sada se nećemo koncentrirati na culling objekata, već ćemo samo pogledati culling pojedinačnih trouglova



Culling

- Postoje tri uobičajena razloga za provedbu procesa cullinga određenog trougla
 - Ukoliko trougao ne leži unutar volumena pogleda (odbacivanje iz frustum pogleda – *view frustum culling*)
 - Ukoliko je trougao okrenut "od" posmatrača (uklanjanje stražnjih poligona – *backface culling*)
 - Ukoliko je trougao *degeneriran* (površina=0)
- Prvi slučaj je automatski ugrađen u algoritam odsijecanja o kojem smo već govorili

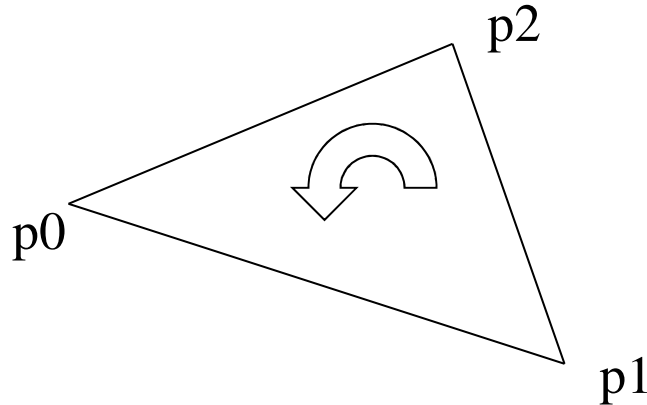


Backface Culling

- Vrlo čest je slučaj da se trouglovi koriste za modeliranje *površine* nekog objekta koji je po prirodi inherentno volumetrijski
- U mnogim slučajevima, većina ili svi ovi trouglovi će biti vidljivi samo sa spoljne strane objekta
- U određenom smislu, ovi trouglovi su jednostrani i jedino vidljivi "sprijeda", pošto ih nikada nećemo morati gledati "odzada"
- Svaki stražnji trougao (engl. *back facing triangle*) treba biti što prije odbačen, pošto je za očekivati da će do 50% trouglova u sceni biti stražnji trouglovi
- Uklanjanje stražnjih poligona (engl. *backface culling*) se obično vrši prije odsijecanja, pošto je to vrlo brza operacija i uticat će na mnogo veći procent trouglova nego odsijecanje

Backface Culling

- Po konvenciji, lice trougla se definira kao strana gdje su vrhovi aranžirani na način "suprotno kretanju kazaljke na satu"



- Većina renderera dozvoljava da trouglovi budu definirani kao jednostrani ili dvostrani. Samo jednostrani trouglovi moraju biti odbačeni
- Također, rendereri obično dozvoljavaju korisniku i da specificira da li će biti odbačeno lice ili naličje, budući da postoje razni slučajevi u kojima korisnik može htjeti jednu ili drugu stranu trougla (određeni specijalni efekti, slučajevi kada se koriste refleksijske transformacije...)

Backface Culling

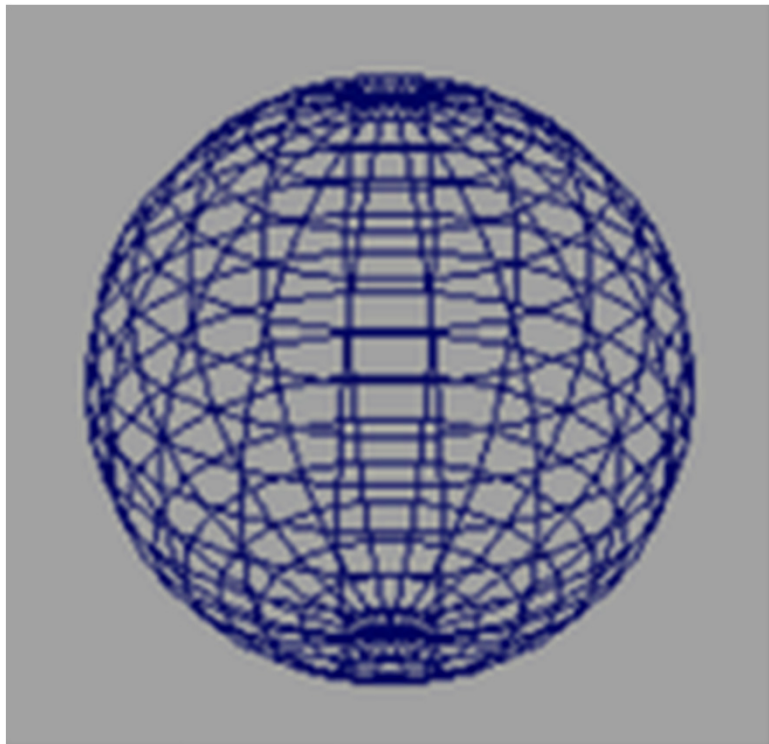
- Uklanjanje stražnjih poligona obično se obavlja u 3D prostoru kamere, pri čemu je sama kamera locirana u (0,0,0). Međutim, uklanjanje stražnjih poligona može biti urađeno i u prostoru objekta (čak prije transformacije) transformiranjem pozicije kamere u prostor objekta
- Prvo se izračunava normala trougla (normala ne mora biti jedinične dužine)
- Zatim se provjerava da li je normala otklonjena više od 90 stepeni od vektora od trougla do položaja kamere (\mathbf{e})

$$\mathbf{n} = (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0)$$

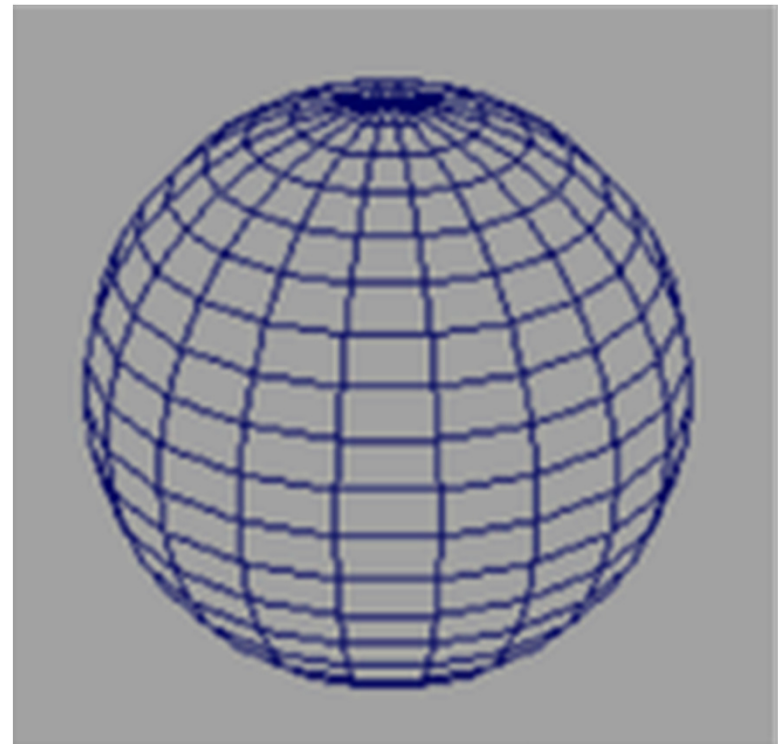
ako je $(\mathbf{e} - \mathbf{p}_0) \cdot \mathbf{n} \leq 0$ tada je trougao nevidljiv



Sa stražnjim poligonima



Bez stražnjih poligona




Degenerirani culling

- Također, bilo bi zgodno i kada bismo na ovom mjestu mogli odbaciti bilo koji *degenerirani* trougao
- Degenerirani trougao ima svoja tri vrha raspoređena na takav način koji uzrokuje da površina trougla bude 0
- Ovo bi se moglo desiti ako sva 3 vrha leže na jednom pravcu
- To bi se također moglo desiti i ako se 2 vrha (ili pak sva 3) nalaze na tačno istom mjestu
- Nasreću, *backface cull* test će automatski odbaciti ovaj slučaj, pošto će u tim slučajevima \mathbf{n} biti $[0\ 0\ 0]$

$$\mathbf{n} = (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0)$$

ako je $((\mathbf{e} - \mathbf{p}_0) \cdot \mathbf{n} \leq 0)$ tada je trougao nevidljiv



Transformacija, projekcija, odsijecanje

- U ovom trenutku smo vidjeli kako uzeti skup trouglova definiran u prostoru objekta i transformirati/projektovati/odsjeći/odbaciti ih u novi skup trouglova u prostoru uređaja, koji se garantirano nalazi unutar granica viewpota
- U postupku odsijecanja/cullinga, neki trouglovi će biti odbačeni, a neki trouglovi će biti kreirani
- Najjednostavniji način da se to uradi je obrađivati jedan po jedan trougao
- Složenije implementacije mogu pokušati da rade nad skupom trouglova istovremeno kako bi podijelile neki posao, ali se sada nećemo fokusirati na ovo pitanje



Scan konverzija

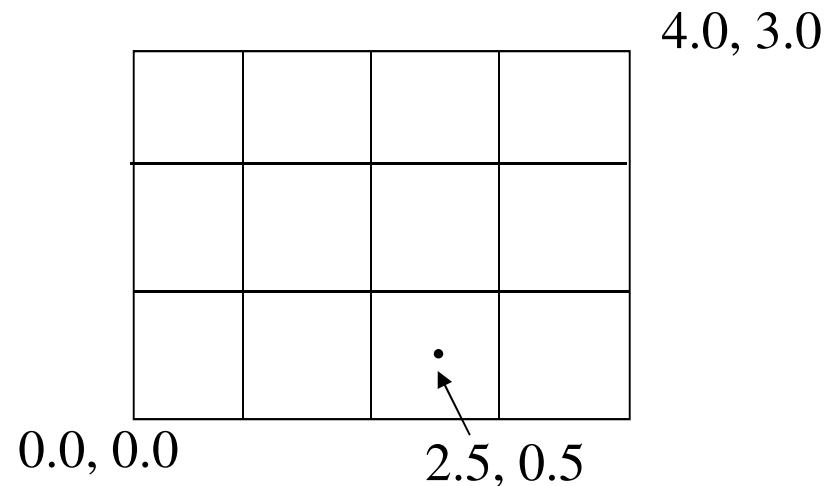


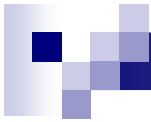
Scan konverzija

- Sada kada imamo 2D trouglove u koordinatama uređaja, moramo egzaktno odrediti koje piksele ti trouglovi pokrivaju
- Ovaj postupak je poznat kao *scan konverzija* ili *rasterizacija*
- Scan konverzija je zapravo 2D postupak koji djeluje u xy prostoru uređaja
- U scan konverziji, podaci "po vrhu", poput boje i dubine, se interpoliraju preko poligona tako da se jedinstvena boja i dubina izračunavaju po pikselu
- Također, uobičajeno je da se interpoliraju i drugi "po vrhu" podaci kao što su koordinate texture, normale ili proizvoljna svojstva specifična za aplikaciju

Koordinate uređaja

- Ako imamo viewport s 800 x 600 piksela, koordinate uređaja su u intervalu od 0.0 do 800.0 po x i od 0.0 do 600.0 po y
- Centar donjeg lijevog piksela je 0.5, 0.5, a centar gornjeg desnog piksela je 799.5, 599.5





Rasterizacija trougla

- Glavni fokus će biti stavljen na rasterizaciju trougla, ali važno je napomenuti da se i ostali primitivi renderinga, kao što su linije i tačke, također mogu rasterizirati



Pravila rasterizacije

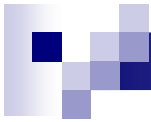
- Osnovno pravilo za rasteriziranje trougla je da želimo da piksel bude ispunjen ako se centar tog piksela nalazi unutar trougla
- Ukoliko centar piksela tačno dodiruje brid ili vrh trougla, piksel ćemo renderirati samo ako trougao pokriva tačku (a ne piksel) neposredno desno od centra piksela (poput 0.000001 s desna)
- Definiranjem ovakvih preciznih pravila renderiraju se tačno isti pikseli bez obzira na redoslijed kojim se renderiraju trouglovi (također, niti jedan piksel ne biva dvaput renderiran ukoliko se nalazi na granici dva ili više trouglova)
- Također, važno je napomenuti da su koordinate vrhova u prostoru uređaja uvijek *floating-point* vrijednosti i *ne trebaju* biti zaokružene na cijele vrijednosti





Rasterizacija trougla

- Postoji nekoliko različitih pristupa rasterizaciji trougla koji su godinama razvijani kako bi se zadovoljili različiti sistemi
- Moguće ih je grubo klasificirati kao *sekvencijalne* i *paralelne* pristupe
- Sekvencijalni pristupi ispunjavaju piksele po redu i iskorištavaju brze inkrementalne izračune
- Paralelni pristupi zahtijevaju potpuniji izračun, ali mogu biti raspoređeni na nekoliko različitih proračunskih jedinica (unutar jednog čipa)
- Paralelni pristupi obično imaju prednost i bržeg setupa, te mogu biti efikasniji za vrlo male trouglove (kao što su trouglovi koji pokrivaju 3 piksela ili manje). Ovo je zapravo vrlo korisno za rendering izrazito teseliranih površina
- Mi ćemo se uglavnom usredotočiti na sekvencijalne metode, pošto su one uobičajenije u modernim sistemima i uvijek mogu biti paralelizirane u određenoj mjeri



Sekvencijalna rasterizacija trougla

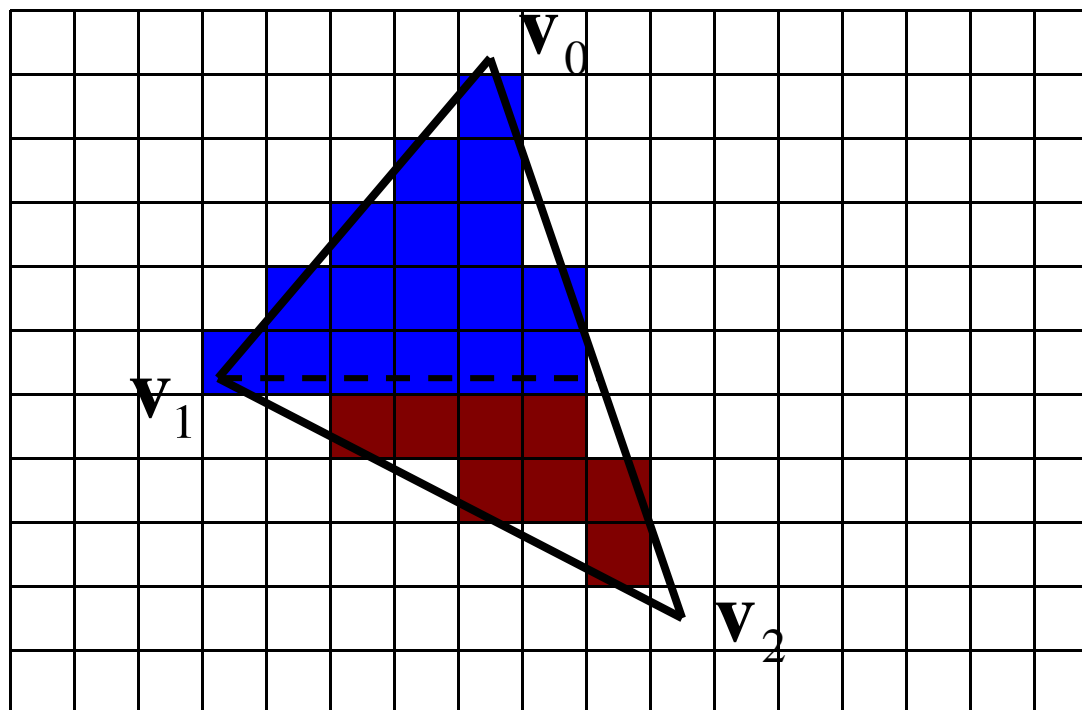
- Počet ćemo na vrhu i spuštati se jednu po jednu *scan liniju*
- Za svaku scan liniju vršit ćemo ispunu slijeva nadesno
- To je najčešći način na koji ljudi o ovome razmišljaju, mada različiti rendereri ovo mogu raditi u bilo kojem redoslijedu
- U hardverskom rendereru, redoslijed može biti aranžiran na takav način da se optimizira performansa pristupa memoriji



Rasterizacija trougla

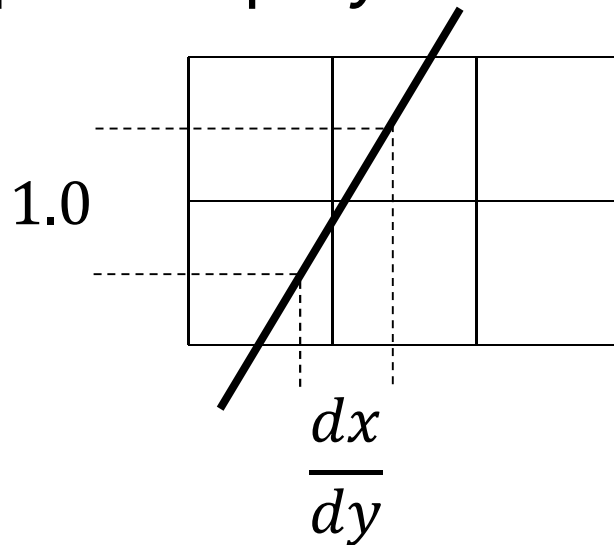
- Ulaz u postupak rasterizacije trougla su tri 2D vrha, svaki s vrijednošću dubine i mogućim drugim svojstvima
- Prvi korak je sortirati ove vrhove u redoslijedu odozgo prema dolje
- Pretpostavit ćemo da je \mathbf{v}_0 najviša tačka (najveća y vrijednost), sa \mathbf{v}_1 u sredini a \mathbf{v}_2 na dnu
- Ovo će možda biti protivno usvojenom *counterclockwise* redoslijedu, ali to na ovom mjestu i nije bitno, pošto se ta informacija koristi pri *backface cullingu* koji se već trebao dogoditi do ovog trenutka
- Postupak će biti da prvo renderiramo gornju polovinu trougla (dio između \mathbf{v}_0 i \mathbf{v}_1), a zatim donju polovinu (od \mathbf{v}_1 do \mathbf{v}_2)

Rasterizacija trougla



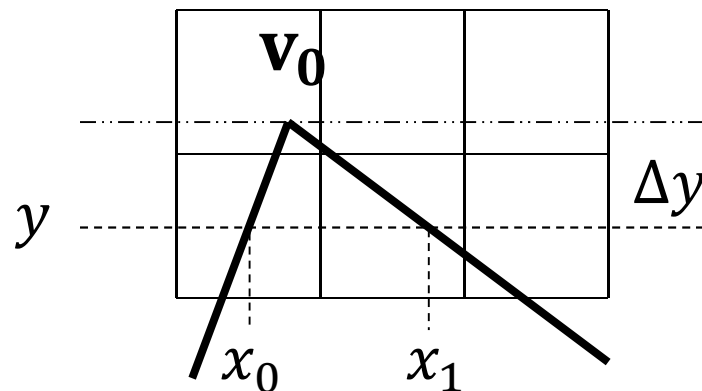
Proračun nagiba

- Da bismo inicijalizirali rasterizaciju, prvo moramo izračunati nagib svake stranice trougla
- Izračunat ćemo dx/dy tako da znamo koliko se x vrijednost mijenja sa svakim punim korakom piksela po y



Određivanje prve scan linije

- Pretpostavićemo da popunjavamo gornju polovinu trougla (od \mathbf{v}_0 do \mathbf{v}_1). Da bismo započeli postupak, moramo odrediti prvu aktualnu scan liniju
- Također, određujemo i x vrijednosti gdje stranice $\mathbf{v}_0\mathbf{v}_1$ i $\mathbf{v}_0\mathbf{v}_2$ sijeku tu scan liniju
- Njih ćemo označiti kao x_0 (lijevo) i x_1 (desno)
- Da bi se to brzo uradilo, mogu se upotrijebiti izračunati nagibi



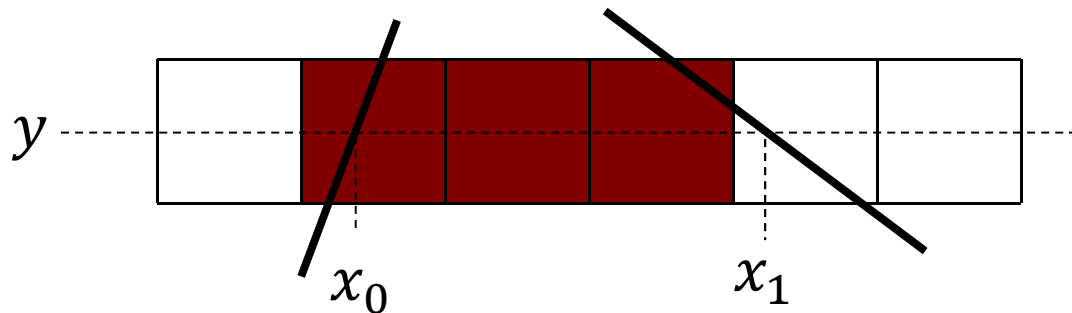
$$y = \text{floor}(v_{0y} + 0.5) - 0.5$$

$$\Delta y = \text{frac}(v_{0y} + 0.5)$$

$$x_0 = v_{0x} - \Delta y \frac{dx_0}{dy}$$

Popunjavanje razmaka

- Sada se petlja može izvesti od x_0 do x_1 da bi se ispunili stvarni pikseli, vodeći računa o tome da se ispunjavaju samo pikseli čiji se centri nalaze između x_0 i x_1



Izvođenje petlje po Y

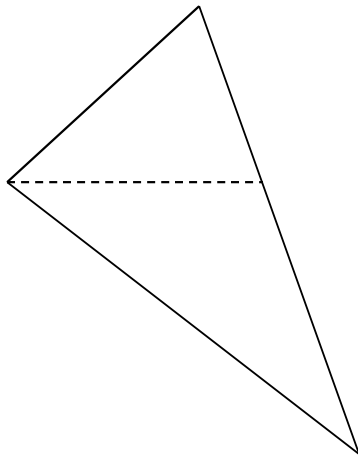
- Petlja se izvodi preko svih scan linija od \mathbf{v}_0 prema \mathbf{v}_1
- Za svaku scan liniju računaju se nove vrijednosti za x_0 i x_1 inkrementalno pomoću nagiba

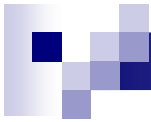
$$x_0 = x_0 - \frac{dx_0}{dy}$$

$$x_1 = x_1 - \frac{dx_1}{dy}$$

Izvođenje petlje po Y

- Petlja se izvodi od \mathbf{v}_0 prema \mathbf{v}_1
- Nakon toga, ponovo se izračunava nekoliko nagiba i započinje rendering donje polovine trougla od \mathbf{v}_1 prema \mathbf{v}_2 , koji se obavlja istim postupkom kao i za gornju polovinu





Sekvencijalna rasterizacija

- Kao što se može vidjeti, stvarni trošak po pikselu je vrlo mali, a i trošak po scan liniji također je vrlo mali
- Međutim, postoji nekakav trošak u setupu (nekoliko operacija dijeljenja i neke druge stvari)
- Algoritam dobija na brzini korištenjem inkrementalnog izračuna većine potrebnih podataka pomoću jednostavnih operacija sabiranja
- U hardveru se čitav postupak rasterizacije obično obavlja s aritmetikom fiksnog zareza, a mnogo posla se može uraditi s fiksnim zarezom prilično male preciznosti (16 bitova je obično dovoljno za većinu operacija)

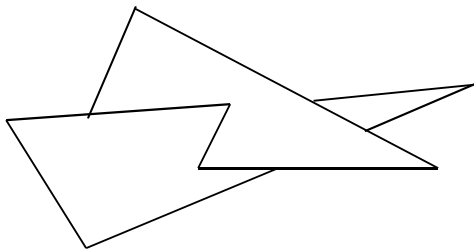


Uklanjanje skrivenih površina

- Termin *uklanjanje skrivenih površina* (engl. *hidden surface removal*) odnosi se na postupak kojim se osigurava da pikseli trouglova blokirani od strane drugih piksela ne budu iscrtani
- Historijski gledano, postojalo je nekoliko pristupa ovoj problematici, ali je danas najpopularniji pristup (unutar "tradicionalnog grafičkog cjevovoda") upotreba z-buffera

Z-buffer

- Kod z-buffer tehnike, svaki piksel pohranjuje vrijednost dubine (ili z-vrijednost), često u 32-bitnom formatu fiksnog zareza
- Kada se ekran "očisti" na početku renderinga jednog frame-a, očisti se i z-buffer, a svaka vrijednost u bufferu se postavlja na "najdalju" vrijednost
- Kada se trougao rasterizira, z vrijednost se interpolira preko trougla da bi se izračunala z vrijednost za svaki piksel
- Prije nego što se piksel renderira, interpolirana z vrijednost se upoređuje s vrijednošću pohranjenom u z-bufferu za taj piksel
- Piksel se renderira samo ako je "bliži" od vrijednosti koja je već u z-bufferu
- Kada se piksel renderira, nova (bliža) z vrijednost se zapisuje u framebuffer
- Ovaj postupak osigurava precizno uklanjanje skrivenih površina tako da će se trouglovi ispravno renderirati čak i kada presijecaju druge trouglove



Z-buffer

- Velika prednost z-buffera je što ne trebamo voditi računa kojim redoslijedom iscrtavamo
- Primjer iscrtanog automobila u OpenGL-u bez i sa upotrebom z-buffera





Z interpolacija

- Da bi se implementirao z-buffering, mogu se napraviti neki relativno minorni dodaci postupku scan konverzije
- Budući da je trougao ravan, na početku rasterizacije mogu se izračunati dz/dy i dz/dx vrijednosti za cijeli trougao
- z vrijednost se inkrementira gotovo na isti način kako su prethodno bile povećavane i x vrijednosti
- Ovo u velikoj mjeri funkcionira zato što se bira perspektivno preslikavanje koje čuva pravce



Ograničenja z-buffera

- Z-buffering je odlična, jednostavna, brza tehnika, ali se mogu povremeno pojavljivati vizualni artefakti ako se ne postupa pažljivo
- Kao prvo, perspektivno preslikavanje uzrokuje da z-buffer ima veću rezoluciju za bliže objekte nego za udaljene objekte
- Ovo je zapravo dobra stvar, jer obično detaljnije renderiramo objekte kada su blizu i manje detaljno kada su udaljeni. Objekti s više detalja zahtijevaju veću preciznost u z-bufferu
- Uobičajen problem kod z-bufferinga je tzv. z-nadmetanje (engl. *z-fighting*)
- Kada su dva trougla koplanarni ili blisko koplanarni, z interpolacija pomoću aritmetike fiksnog zarezca može prouzrokovati greške i vidjet ćemo titrajuće uzorke ondje gdje se dva trougla preklapaju
- Postoji nekoliko načina da se to popravi, a jedan od načina da se preciznost poboljša je da omjer dalje/bliže rastojanje odsijecanja bude što manji
- Naprimjer, bliže rastojanje odsijecanja od 0.1 i dalje rastojanje odsijecanja od 1000 (omjer 10000) radi relativno dobro sa 32-bitnim z-bufferom. Veći omjeri mogu prouzrokovati više problema...



Interpolacija svojstava

- Osim interpoliranja z vrijednosti preko trougla, mogu se interpolirati i druga svojstva
- Primjeri uključuju boju, normale, koordinate texture ili druge stvari
- Naprimjer, da bismo interpolirali boju, crvena, zelena i plava komponenta biće interpolirane na isti način kao i z vrijednost
- Kada budemo razmatrali preslikavanje texture, vidjet ćemo da je potrebno obaviti određen dodatni posao kako bi se osiguralo da svojstva budu ispravno interpolirana