1. OSNOVE

Sceno predstavljamo s **trikotniki**, uporabljamo tudi **normale**. Točke obstajajo samo v koordinatnih sistemih, vektorji pa v vektorskih prostorih. Če želimo enotski vektor, ga moramo normirati: $\vec{v_n} = \frac{\vec{v}}{|\vec{v}|}$.
Evklidski koordinatni sistem vsebuje bazne vektorje z dolžino 1. Ti so pravokotni drug na drugega. V R^3 imamo dve možnosti za postavitev baznih vektorjev: **levosučni** (DirectX) in **desnosučni** (OpenGL) k.s.

Vrednost (skalar) med dvema vektorjema nam pove **kot** med vektorjema in **dolžino projekcije** na enotski vektor. Skalarni produkt ima veliko uporab:

- pri računanje osvetlitve nas zanima kot vpadne svetlobe (večji kot → manj osvetljeno)
- pri detekciji trkov nas zanima v katero smer se predmet premika relativno na drug predmet, pod kakšnim kotom se zaletita
- razdalja od točke do ravnine
- katera stran poligona gleda proti kameri
- pretvorbe med koordinatnimi sistemi

Vektorski produkt je definiran le v 3D prostoru kot $\vec{c} = \vec{a} \times \vec{b}$. Dolžina je $|\vec{a} \times \vec{b}| = |\vec{a}||\vec{b}| \sin\theta$. Rezultat je vektor, ki je **pravokoten** na $\vec{a}$ in $\vec{b}$ v **desnosučnem** k.s.

$$\vec{a} \times \vec{b} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{bmatrix}$$

Pomemben je pri iskanju **normal**. Izračunamo jo kot: $\vec{n} = (b - a) \times (c - a)$, kjer so a, b, c oglišča trikotnika.

1.1 Transformacije

Predmete predstavimo z **množico točk**. Transformacije preslikajo eno konfiguracijo točk v drugo, s tem spremenijo položaj, usmeritev, velikost in obliko predmetov. Predmeti imajo položaj, usmeritev in velikost v prostoru, določamo jih s translacijo, rotacijo in skaliranjem. Kamera, ki gleda na predmete, ima položaj in usmiritev - translacija in rotacija. Pri projeciranju 3D točk na 2D projekcijsko ravnino uporabimo planarno projekcijo.

Transformacija (preslikava) je funkcija, ki vzame točko ali vektor in jo preslika v neko drugo točko ali vektor. Afine transformacije ohranjajo vzporedne črte, projekcijske transformacije pa ohranjajo le črte.

1.1.1 Afine 2D transformacije

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c \\ f \end{bmatrix}$$

$$p' = Mp + t$$

Translacija:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} dx \\ dy \end{bmatrix}$$

Skaliranje (poteka okoli središča; če je s_x različen od s_y, dobimo neenakomerno skaliranje):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Skaliranje okoli poljubne točke: transformacije sestavimo:

- premik točke skaliranja v središče k.s.
- skaliranje
- potem premik nazaj

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} (\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} dx \\ dy \end{bmatrix}) + \begin{bmatrix} dx \\ dy \end{bmatrix}$$

Rotacija okoli središča k.s. je ekvivalentna rotaciji k.s v obratni smeri (pozitivni koti so v nasprotni smeri urinega kazalca):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Če rotacijo izvedemo okoli poljubne točke, sestavimo transformacije kot pri skaliranju.

Težava pri transformacijah nastane, ker imamo heterogene operacije (množenje in seštevanje), želimo pa si le homogene operacije, tako da lahko transformacije združujemo.

Homogene koordinate so matematični konstrukt, ki omogoča, da vse afine transformacije pretvorimo v množenje matrik in jih tako enostavno sestavljamo. Pri tem uvedemo dodatno koordinato za zapis točk (za točke postavimo $w = 1$, za vektorje $w = 0$; projekcije spreminjajo w). Pretvorbo iz točk v homogenih koordinatah v običajne koordinate dobimo z deljenjem. Bistvo vsega je, da **translacija postane množenje**. Pri skaliranju in rotaciji dodamo eno vrstico/stolpec.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homogene koordinate nam omogočajo enostavno sestavljanje transformacij,pri tem je važen vrstni red množenja matrik (od desne proti levi).

Poznamo še dve 2D transformaciji:

- **zrcaljenje**: zrcalna slika predmetov, običajno poteka preko neke osi; če so osi kar osi k.s. potem govorimo o spremembi predznaka x ali y koordinat predmeta
- **striženje**: raztegne predmete v odvisnosti od oddaljenosti od osi; h je faktor striženja v smeri x, g v smeri y
  - $\begin{bmatrix} 1 & h & 0 \\ g & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

1.1.2 Afine 3D transformacije

Uvedemo 3. dimenzijo in še vedno delamo s homogenimi koordinatami. Zaplete se pri rotaciji:

- x: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- y: $\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- z: $\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Eulerjevi koti predstavljajo kote rotacij okrog koordinatnih osi, da dosežemo želeno usmeritev. Do problema pride, če se dve osi vrtenja poravnata med seboj. Temu pravimo **kardanska zapora**. To pomeni izgubo prostostne stopnje, saj se predmet ne bo vrtel, kot smo si zamislili. Rotacijo lahko zato predstavimo z osjo okoli katere je predmet rotiran, in kotom rotacije. Pri animacijah si lahko pri tovrstni predstavitvi pomagamo s **kvaternioni**. To so 4D kompleksna števila.

Sestavljene transformacije predstavimo z množenjem matrik. Tipična sestava osnovnih treh transformacij je TRS (translacija, rotacija, skaliranje).

Normale so definirane kot vektor pravokoten na drug vektor oz. ploskev, ne transformiarjo se na enak način kot točke. $n \cdot v = 0$, $(Xn)^T(Mv) = 0$. Če točko transformiramo z M, normaler transformirano z $(M^{-1})^T$.

1.2 Koordinatni sistemi

3D scena je sestavljena iz predmetov, postavljenih v prostor - svet. Na svet gledamo skozi kamero, ki se tudi nahaja nekje v prostoru.

Posamezen predmet je narejen relativno glede na svoj **lokalni koordinatni sistem**. Ta je ponavadi v središču predmeta; predmet ima tudi lokalne koordinate. Predmeti so preko **transformacije modela** postavljeni v svet. Točke se iz lokalnih koordinat transformirajo v koordinate sveta.

Na svet gleda kamera, ki je na nekem položaju in orientaciji v svetu. Slika, ki jo vidimo, če pogledamo skozi kamero, se imenuje **projekcijska ravnina**. Predmete iz koordinat sveta v koordinate pogleda preslikamo s transformacijo pogleda. Položaj kamere je določen s transformacijo kamere $T_k R_k$, transformacija pogleda pa je obratna transformacija kamere $R_k^{-1} T_k^{-1}$.

Koordinatni sistem kamere določimo:

- $n = \frac{g-c}{|g-c|}$; n kaže v smeri pogleda
- $u = \frac{n \times y}{|n \times y|}$; u kaže desno
- $v = \frac{u \times n}{|u \times n|}$; v kaže gor

Predmet, ki je predstavljen z matriko točk v lokalnih koordinatah, preslikamo v koordinate pogleda.

2. PROJEKCIJE

Predmete preslikamo: lokalne koordinate → koordinate sveta → koordinate pogleda / kamere.

Planarna projekcija je preslikava iz 3D predstavitve predmeta na projekcijsko ravnino (2D). Take preslikave ohranjajo črte, ne pa nujno tudi kotov. **Neplanarne projekcije** se uporabljajo npr. pri zemljevidih. Planaerne projekcije delimo na:

- **vzporedne**: projekcijski žarki so vzporedni
- **perspektivne**: projekcijski žarki konvergirajo v točko

2.1 Vzporedna projekcija

Imenovane tudi kot **pravokotne (ortografske) projekcije**. Projekcijski žarki so **pravokotni** na projekcijsko ravnino, zato predmeti daleč izgledajo enako veliki kot tisti blizu (ni efektov perspektive). Glede na položaj kamere lahko ločimo različne poglede:

- projekcijska ravnina je **vzporedna** z osnovnimi pogledi na predmet (tloris, naris, stranski ris; ohranja dolžine stranic in kote)
- projekcijska ravnina je **poševna** glede na predmet (ločimo glede na število kotov, ki so na projecirani kocki - izometrična (3), dimetrična (2), trimetrična (0); razmerja dolžin črt se ohranjajo, koti se ne).

V koordinatah pogleda sta x in y poravnana z ravnino projekcije, z pa pravokotno kaže v 3D sceno ali stran (lahko zanemarimo z). Pri aksonometričnih projekcijah dodamo še ustrezno rotacijo kamere.

Pravokotne projekcije uporabljamo pri CAD, tehničnih risbah, ilustracijah, za natančen prikaz predmeta. Pri igrah je to koristno, ker so oddaljeni predmeti enako veliki kot bližnji - vidimo tudi podrobnost oddaljenih predmetov.

2.2 Perspektivna projekcija

Projekcijski žarki se stikajo v točki. Črte, ki so na predmetu vzporedne, in niso na ravnini, ki je vzporedna projekcijski ravnini, se sekajo v **ponornih točkah**. Če predpostavimo, da so predmeti preslikani v k.s. kamere in da je d razdalja od kamere do projekcijske ravnine, dobimo perspektivno matriko:

$$\begin{bmatrix} xd/z \\ yd/z \\ d \\ 1 \end{bmatrix}$$

Perspektivne poglede lahko glede na postavitev kamere delimo na število ponornih točk (eno, dvo ali tritočkovna perspektiva). Uporabljamo jo, ko želimo realističen pogled na predmet (manjši predmeti tisti ki so dlje), sicer dobimo popačene predmete.

Razdalja med projekcijsko ravnino ter gledališčem in velikost izseka na projekcijski ravnini določa **zorni kot** $FOV\alpha = 2\tan\frac{h}{2d}$.

Tipični parametri navidezne kamere:

- položaj in orientacija
- slikovno razmerje
- zorni kot
- prednja in zadnja ravnina rezanja (določata vidno prisekano piramido; le predmeti znotraj te piramide so vidni)
- fokusna razdalja (simulacija globinske ostrine)

3. GRAF SCENE

Predmeti/scene so tipično sestavljeni iz več delov, lahko bi jih predstavili neodvisno s seznamom delov (vsak ima svoje lastnosti in transformacije). Vendar deli v sceni med seboj velikokrat niso neodvisni. Rešitev se skriva v združevanju predmetov v skupine. Vsaka skupina ima seznam predmetov in si deli lastnosti (hierarhija) → dobimo drevo. Povezave nosijo podatke o relativnem položaju glede na starša (transformacijske matrike). Pri izrisu predmeta je transformacijska matrika produkt matrik na poti od korena do predmeta.

En predmet je lahko v sceni večkrat izrisan, vsak ima svojo transformacijsko matriko. Transformacije se ravno tako akumulirajo na poti od korena do predmeta. Za izris grafa scene uporabljamo sklad (rekurziven sprehod). Predmeti imajo poleg položaja še druge lastnosti - barva, material, animacijski parametri . . . , ki jih tudi dodamo v graf.

Hierarhija predmetov omogoča tudi lažje:

- hierarhično deljenje prostora
- izločanje predmetov, ki jih ne vidimo še pred izrisom
- izračun nivojev podrobnosti
- urejanje poligonov od zadnjega proti prvem
- itn.

4. BARVANJE

Transformacija v k.s. pogleda in projekcija potekajo v **senčilniku ogljišč**. Rezultat so trikotniki na projekcijski ravnini.

Rasterizacija določi, katere piskle/fragmente porkiva vsak trikotnik in interpolila vrednosti v ogliščih na nivo piksla/framenta. Določanje barve pikslov/fragmentov poteka v **senčilniku fragmentov**. Barvo določimo glede na:

- barvo v ogliščih
- teksturo (v 2D tipično slika, v 3D zaporedje slik ali proceduralna tektura)

4.1 Interpolacija

Z **interpolacijo** določimo vrednost v notranjosti trikotnika na podlagi vrednosti v ogliščih. Gre za kakršnekoli lastnosti - barva, normale, teksture koordinat, globina ipd.

Pri **bilinearni interpolaciji** linearno interpolacijo izvedemo najprej po eni, nato po drugi osi. Postopek interpoliranja:

- trikotnik z oglišči A, B, C in vrednostmi V_A, V_B, V_C, ki jih želimo interpolirati v točki P
- najprej interpoliramo po eni osi (npr. y), da dobimo vrednosti v Q in R
  - $V_Q = \frac{y_P - y_B}{y_A - y_B} V_A + \frac{y_A - y_P}{y_A - y_B} V_B$
  - $V_R = \frac{y_P - y_B}{y_C - y_B} V_C + \frac{y_C - y_P}{y_C - y_B} V_B$
- potem interpoliramo po osi x, da dobimo končno vrednost v P
  - $V_P = \frac{x_C - x_P}{x_C - x_B} V_Q + \frac{x_P - x_B}{x_C - x_B} V_R$
- izračun v zaporednih točkah lahko pohitrimo

Pri trikotnikih lahko definiramo tudi t.i **težiščni** (baricentrični) koordinatni sistem - koordinatni osi sta stranici trikotnika. Točko v tem prostoru predstavimo kot:

- $P = A + \beta(B - A) + \gamma(C - A)$ oz.
- $P = \alpha A + \beta B + \gamma C$
- $\alpha + \beta + \gamma = 1$

V 2D prostoru lahko zapišemo:

- $\beta(B - A) + \gamma(C - A) = P - A$
- $\begin{bmatrix} x_B - x_A & x_C - x_A \\ y_B - y_A & y_C - y_A \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} x_P - x_A \\ y_P - y_A \end{bmatrix}$

in dobimo:

- $\beta = \frac{(x_A - x_C)(y_P - y_C) - (y_A - y_C)(x_P - x_C)}{(x_B - x_A)(y_C - y_A) - (x_C - x_A)(y_B - y_A)}$
- $\gamma = \frac{(x_B - x_A)(y_P - y_A) - (y_A - y_B)(x_P - x_A)}{(x_B - x_A)(y_C - y_A) - (x_C - x_A)(y_B - y_A)}$
- $\alpha = 1 - \beta - \gamma$

Če poznamo predstavitev točke P s težiščnimi koordinatami α, β, γ lahko:

- enostavno izračunamo **bilinearno interpolacijo** lastnosti V oglišč v točki: $V_P = \alpha V_A + \beta V_B + \gamma V_C$
- enostavno ugotovimo, če je točka znotraj trikotnika:
  - $0 < \alpha, \beta, \gamma < 1$: točka je znotraj trikotnika
  - $0 \leq \alpha, \beta, \gamma \leq 1$: točka je znotraj ali na robu trikotnika
  - sicer je točka izven trikotnika

Če uporabljamo perspektivno projekcijo, ta popači predmete (ni linearna, delimo s homogeno koordinato w). Navadna bilinearna interpolacija po projekciji ne upošteva pravilno tega popačenja. Pri tem interpolirane vrednost homogeniziramo s faktorjem $w = \frac{z}{d}$, nato izvedemo bilinearno interpolacijo homogeniziranih vrednosti. Bilinearno interpoliramo tudi faktorje, s katerimi smo množili, nato pa nazaj popravimo interpolirane homogenizirane vrednosti.

Interpolacija lastnosti oglišč na posamezne fragmente se izvede med **rasterizacijo**. Spremenljivke **interpoliranke**, ki jih z **out-in** prenesemo med senčilnikom oglišč in fragmentov, se **avtomatsko** interpolirajo.

4.2 Teksture

Enobravni poligoni so enostavni, hitri in nezanimivi. Ker želimo več podrobnosti, na ploskve nalepimo teksture. Tekstura lahko vpliva na barvo, pa tudi na druge elemente: normale, položaj oglišč (simulacija materialov, zmanjševanje geometrijske kompleksnosti, odsevi).

4.2.1 2D teksture

Bitna slika (2D tekstura) se na predmet preslika s pomočjo **uv koordinatnega prostora** teksture: pikslom teksture (texels) določimo koordinate u, v na območju $[0, 1]$ - levo spodaj je $(0, 0)$, desno zgoraj $(1, 1)$. Vsako oglišče trikotnika hrani u, v koordinate dela teksture, ki se nanj preslika. Če u, v koordinare padejo izven tega območja:

- teksturo **ponavljamo** ali **zrcalimo**
- uporabi **barvo roba teksture** ali teksture ne upoštevamo

Za vsako točko v trikotniku lahko z **interpolacijo** uv koordinat izračunamo, v katero točko teksture se preslika. Interpolacija je standardna **perspektivno pravilna bilinearna** interpolacija (uv koordinate oglišč prenesemo v senčilnik fragmentov v out-in interpolirankah). Dobljene interpolirane u, v koordinare navadnno ne padejo "na"

teksel. Barvo na u, v izračunamo s metodo **najbližji sosed** (hitra, slaba kvaliteta) ali pa z **bilinearno interpolacijo** (najprej izračunamo c_t in c_b, potem c).

Če uporabimo perspektivno pravilno interpolacijo in bilinearno iterpolacijo lahko pride do **prekrivanja - aliasing**. Na teksturo gledamo kot na signal; če vzorčimo dovolj pogosto, je vse vredu, če pa ne, hitro menjavanje barv postane počasno menjavanje barv, temu pa rečemo **prekrivanje**. Do tega pride, ker lahko piksel na sliki pokrije **velik kos teksture**. Izognemo se ga lahko s **povprečenjem tekslov**.

Povprečenje izvedemo vnaprej - **mipmapping**: vnaprej izračunamo več verzij teksture različnih velikosti - mipmaps:

1. izračunamo koordinate u, v
2. izračunamo približno **velikost piksla** v teksturnem prostoru
3. barvo z **bilinearno interpolacijo** izberemo iz ustrezno velike teksture

Za bolj mehek prehod med nivoji lahko uporabimo dva nivoja - **trilinearna interpolacija**.

Obstaja še en način izogibanja prekrivanju - anizotropično filtriranje. Medtem ko mipmapping povpreči teksturo v vseh smerek, kot da je piskel kvadrat (če gledamo pod kotom ni kvadrat), anizotropično filtriranje povpreči teksturo v tekslih znotraj okna, izračunanega za vsak piksel:

- okno je odvisnmo od orientacije ploskve in je boljši približek realnosti
- povprečimo določeno število tekslov znotraj tega okna
- boljša kvaliteta od mipmappinga, bolj zahtevno, predvsem glede pretoka podatkov

4.2.2 Lepljenje tekstur

Teksturo lahko projeciramo na 3D model in izračunamo u, v koordinate. Lepljenje delamo ročno z 3D modelom preslikanim v u, v prostor.

- vzporedno (planarno) lepljenje: uporabimo **linearno transformacijo** xyz koordinat predmeta
- perspektivno lepljenje: uporabimo **perpsektivno projekcijo** xyz koordinat predmeta
- sferično lepljenje: uporabimo **sferične koordinate**; kot bi predmet postavili v kroglo in teksturo s krogle nalepili na predmet
  - izračun u,, v v točkjo p, če je krogla z radijem R v središči c:
    - krogelne koordinate p-ja: $\cos\theta = \frac{p_y - c_y}{R}$, $\tan\theta = \frac{p_z - c_z}{p_x - c_x}$
    - u, v koorindate točke p: $u = \frac{\phi}{2\pi}$, $v = \frac{\pi - \theta}{\pi}$
- cilindrično lepljenje: podobno kot sferično, teksturo mapiramo preko valja
  - $\tan\phi = \frac{p_z - c_z}{p_x - c_x}$, $u = \frac{\phi}{2\pi}$
- kožno lepljenje: površino predmeta razvije v 2D - kožp; ročno določimo položaj delov modela v u, v prostoru

4.2.3 3D teksture

Z njimi simulairamo predmet narejen iz nekega materiala. Definirana je v treh dimenzijah. Lahko uporabljamo **neposredno 3D** lepljenje. 3D tekstura je serija slik, velikokrat je specificirana **proceduralno**. Osnova je **funkcija** $f(s,t,r)$, dodamo še šum, da izboljšamo realizem.

Proceduralne teksture navadno vsebujejo komponento šuma. Poznamo **Perlinov šum**, ki je $\mathbb{R}^n$ šumna funkcija. Je **gradientni šum** - interpolacija med naključnimi gradienti, mehki prehodi. Veliko se uporablja, ker je "mehka" funkcija, omogoča enostaven nadzor nad **frekvenco** in **fazo**.

V naravnih teksturah velikokrat srečamo neko regularnost, ki se pojavlja na več velikostnih nivojih. Pravimo ji **turbulenca** = vsota skalarnih Perlinovih šumov.

5. OSVETLJEVANJE

Pri osvetljevanju računamo interakcijo svetlobe s površinami, kar je bolj ali manj kompleksna simulacija fizike. Glede na simulacijo širjenja svetlobe po prostoru lahko ločimo:

- **lokalno** osvetljevanje: je poenostavljeno, saj se upošteva le en odboj med izvorom in gledalcem, računanje osvetlitve ene ploskve poteka neodvisno od ostalih, ne upodabljamo senc, standardno poteka v grafičnem cevovodu

- **globalno** osvetljevanje: upoštevamo več odbojev svetlobe od predmetov, algoritmi bazirajo na sledenju žarkov, je računsko zahtevno in počasi prehaja v uporabo za interaktivne aplikacije

5.1 Modeliranje materialov

Za dobro upodobitev predmetov moramo znat modelirati, kako se svetloba odbija od predmetov. Pri enostavnem modelu je odbita svetloba vsota treh komponent: **razpršeni** odboj, **zrcalni** odboj, **ambientna** svetloba. S tovrstno predstavitvijo lahko zajamemo velik nabor realnih materialov.

5.1.1 Razpršeni odboj

Zgodi se na materialu, ki odbija svetlobo enakomerno na vse strani. Ni važno s katere strani oz. pod kakšnim kotom ga gledamo, izgleda približno enako. Pojavi se na motnih, nesvetlečih materialih, ki so drobno hrapavi na površini.

Odbita svetloba je proporcionalna s kosinusom kota med vpadno svetlobo in normalo na površino. Večji kot je kot, manj svetla je povšina. To pravilo imenujemo **Lambertov kosinusni zakon**.

$$c = c_l k_d (n \cdot l) = c_l k_d \cos\theta$$

Komponente:

- n - normala na površino (normirana)
- l - smer svetlobe
- k_d - razpršena odbojnost - RGB vektor, ki določa, koliko svetlobe se odbije - barva materiala
- c_l - intenziteta vpadne svetlobe - RGB vektor
- c - intenziteta odbiite svetlobe - RGB vektor

V enačbi **ni smeri pogleda** e - od povsod izgleda predmet enako.

5.1.2 Zrcalni odboj

Odblesk/sijaj je zabrisan odsev vira svetlobe, položaj je odvisen od smeri gledanja. Idealen zrcalni odboj je v ogledalu, saj je popolnoma gladka površina, vpadni kod in kot odboja pa sta enaka. Tipični materiali niso popolnoma odbojni. Svetloba se odbija približno v smeri idealnega odboja.

Phongov model: kot α med idealnim odbojem R in smerjo pogleda e določa količino odbite svetlobe; parameter zrcalnega odboja p določa velikost razpršitve - večji p, bližje smo idealnemu odboju.

$$c = c_l k_s (R \cdot e)^p R = 2(l \cdot n)n - l$$

Blinnov model: podoben Phongu, vendar ne potrebujemo izračuna odboja, imamo kompromisni vektor h, ki je na sredini med l in e. Če sta luč in gledalec daleč od površine, lahko predpostavimo, da je h konstanten.

$$c = c_l k_s (h \cdot n)^p h = \frac{l + e}{|l + e|}$$

5.1.3 Ambientna svetloba

V realnosti je del svetlobe povsod, saj se odbija od sten in ostalih predmetov v sceni. Pri lokalni osvetlitvi jo aproksimiramo z ambientno svetlobo - predstavlja približen prispevek svetlobe k splošni sceni ne glede na položaj luči in predmetov. Povsod dodamo konstantno osvetlitev, s tem pa dosežemo, da nimamo več popolnoma temnih delov.

$$c = c_a k_a$$

5.1.4 Izračun osvetlitve in BRDF

Ker imamo lahko več virov svetlobe, seštejemo prispevke vseh. c_a in c_i določajo RGB jakost in barvo svetlobnih virov. k_a, k_d in k_s določajo RGB odbite količine svetlobe snovi - barvo predmeta. Izračunamo s **Phongovim modelom** za več luči:

$$c = c_a k_a + \sum_i c_i (k_d (l_i \cdot n) + k_s (R_i \cdot e)^p)$$

Lambert, Phong in Blinn so nostavni modeli materialov. Posplošen odboj zapišemo s funkcijo **BRDF** - za vsak par: smer vpadne svetlobe (luči) i, in smeri odboja (gledalca) r, določi koliko svetlobe se odbije do gledalca. BRDF je štiridimenzionalna funkcija kotov vpadne in odbite svetlobe. Odboj od luči c_i do opazovalca lahko zapišemo kot: $c = c_i \text{BRDF}(\theta_i, \phi_i, \theta_r, \phi_r) \cos\theta_i$.

5.2 Luči

Luči imajo lahko različne parametre: barvo, površino s katere sevajo svetlobo, usmerjenost, ... Za implementacijo potrebujemo izračun vektorja proti luči l in jakost/barve s katero je točka osvetljena c_l, npr. $c = c_a k_a + c_l(k_d(l \cdot n) + k_s(R \cdot e)^p)$.

Izvor usmerjene luči je zelo daleč (približek sonca). Določata jo barva c_{src} in smer d. Implementacija: $l = -d$, $c_l ? c_{src}$.

Točkasta luč je enostaven model žarnice, ki svetlobo seva v vse smeri enako. Kot vpadne svetlobe je odvisen od položaja p. Jakost pada s kvadratom razdalje. Implementacija: $l = \frac{p-v}{|p-v|}$, $c_l = \frac{c_{src}}{f(|p-v|^2)}$.

Reflektor seva v neko smer d (poleg položaja ima tudi usmeritev). Jakost je odvisna od širine **stožca** θ_{max}, pada pa proti robu stožca s potenco f. Implementacija:

- $l = \frac{p-v}{|p-v|}$
- $c_l = \begin{cases} 0, & \text{če} -L \cdot d \leq \cos(\theta_{max}) \\ c_{src}(-L \cdot d)^f, & sicer \end{cases}$

5.3 Osvetljevanje v grafičnem cevovodu

Implementacija osvetljevanja:

- izberemo model materiala, npr Phongov: $c = c_a k_a + c_l(k_d(l \cdot) + k_s(R \cdot e)^p)$
- glede na izbrani model lahko definiramo lastnosti materiala kot lastnosti oglišč, jih definiramo v senčilniku ali podamo v ločeni teksturi
- za izračun potrebujemo tudi pravilne **normale** v ogliščih
- osvetlitev računamo v ogliščih in za vsak fragment/piksel poligona

Osvetlitev izračunamo v vsakem **oglišču** poligona (torej v senčilniku oglišč ali geometrije).Barvo-svetlost prenesemo v senčilnik fragmentov preko out-in interpolirank. Za izračun osvetlitve potrebujemo **normale** v ogliščih. Za mehke prehode med ploskvami normale postavimo na povprečje normal ploskev, ki se stikajo v oglišču. Gouraudovo senčenje je hitro, kvaliteta je problematična predvsem, ko je število poligonov majhno.

Pri ploskem senčenju je cel poligon enako osvetljen. Računamo na nivoju oglišč, dogaja se v senčilniku oglišč ali geometrije. Implementacija:

- normale oglišč morajo bit enake
- barvo iz senčilnika prenesemo preko interpoliranke *flat out*, da se ne interpolira med ogliščі

Osvetljevanje v fragmentih (tudi Phongovo senčenje) poteka tako, da osvetlitev izračunamo v vsakem pisku/fragmentu poligona (torej v senčilniku fragmentov). Dobimo boljšo kvaliteto in precej mehkejši rezultat kor Gouraud.

6. GRAFIČNI CEVOVOD

Grafični cevovod pretvori predmet/sceno iz računalniškega zapisa v bitno sliko. Sestavljen je iz več faz, ki preslikajo 3D predmete v bitno sliko. Tipično je implementiran v strojni opremi (grafična kartica).

specifikacija oglišč ⇒ senčilnik oglišč ⇒ teselacija ⇒ senčilnik geometrije ⇒ postprocesiranje oglišč ⇒
sestava primitivov ⇒ rasterizacija ⇒ senčilnik fragmentov ⇒ obdelava vzorcev

Primitivi (pšoligoni) se po fazah procesirajo od vhodnih podatkov do izhodne slike. Vsaka faza posreduje rezultate naslednji fazi. Cevovod lahko predstavimo na različne načine. Določene faze so lahko implementirane strojno, druge programsko (senčilniki/shaders).

6.1 Specifikacija oglišč

V tem delu se pripravljajo podatki za upodabljanje. Določimo oglišča in definiramo primitive ter kako se oglišča povežejo v primitive.

6.2 Senčilnik oglišč

Tu poteka obdelava posameznih oglišč. **Program** specificira uporabnik in je obvezen. Tu potekajo tipično transformacije oglišč in normal, izračun osvetlitve in projekcija. Atribute oglišč le spreminjamo, ne dodajamo novih oglišč ali jih brišemo (v senčilnik pride posamezno oglišče z vsemi atrubuti, izhod je prav tako transformirano oglišče z autributi). Atribute, ki jih želimo interpolirati na nivo fragmenta, deklariramo kot out.

6.3 Teselacija

Tu poteka deljenje poligonov na več manjših. Imamo dva senčilnika, ki jih specificira uporabnik: *control* (definira stopnjo deljenja) in *evaluation* (interpolira lastnosti oglišč).

6.4 Senčilnik geometrije

Tu se dodajajo novi primitivi / briše obstoječe, spreminja obstoječe ter pretvarja primitive (npr. točke v trikotnike).

6.5 Postprocesiranje oglišč

Oglišča lahko shranimo nazaj v VBO. Poznamo:

- rezanje
- perspektivno deljenje
- transformacija v koordinate zaslona

6.5.1 Rezanje

Delimo na dve vrsti:

- odstranimo vse ploskve, ki niso v vidnem delu (prisekana piramida)
- porežemo ploskve, ki so delno v, delno pa izven vidnega dela

Režemo v fazi postprocesiranja, lahko pa tudi med rasterizacijo.

Rezanje v 3D je bolj enostavno, če prostor gledanja pretvorimo v **homogene koordinate rezanja** (pretovrimo območje, ki ga kamera vidi, v kocko s stranicami med [-1, 1]). Koordinate rezanja nastanejo pri perspektivni projekciji pred perspektivnim deljenjem. **Koordinate rezanja**: po množenju še **ne** opravimo perspektivnega deljenja.

Režemo torej vse, kar je izven homogenega dela w_c. Odstranimo točke, črte (če so v celoti izven oz. skrajšamo če so delno izven) in trikotnike (če so v celoti izven oz. režemo če so delno izven).

6.5.2 Perspektivno deljenje

Iz koordinat rezanja s perspektivnim deljenjem pretvorimo v normalizirane koordinate naprave - NDC (prisekana piramida pogleda postane kocka med [1, 1, 1] in [-1, -1, -1]; z koordinata ni več linearna).

Iz NDC pretvorimo v koordinate zaslona, ki ustrezajo ločljivosti zaslona. Pretvorba je le ustrezno skaliranje iz NDC.

6.6 Sestava primitivov

Zaporedje oglišč se pretvori v zaporedje primitivov. Izvede se **izločanje zadnjih ploskev**.

6.6.1 Izločanje

Izločiti želimo čimveč predmetov (ali poligonov), ki v upodobitvi ne bodo vidni.

Vrste izločanja:

- **izločanje zadnjih ploskev**: dogaja se na nivoju poligonov v cevovodu (sestava primitivov)
  - če je nastavljeno, da so poligoni **enostranski** - vidni le z ene strani
  - če proti kameri gleda zadnja stran poligona, ga izločimo
  - izločanje je enostavno - glede na vrstni red oglišč pri izrisu trikotnika
- **izločanje predmetov izven prostora pogleda**: dogaja se na nivoju predmetov preden jih pošljemo v cevovod
  - izločimo predmete, ki niso znotraj prostora gledanja
  - ker imajo predmeti kompleksno geometrijo, za hito izločanje uporabimo tehnike razdelitve prostora
  - tovrstne tehnike niso del cevovoda na kartici, ampak jih izveemo že prej (kartici pošljemo le geometrijo, ki je vidna)
- **izločanje zakritih ploskev**: veliko različnih načinov - nekateri za specifične domene, nekateri potrebujejo veliko predprocesiranja
  - koiristno predvsem v primerih, ko je velik del scene zakrit (npr. sprehod skozi mesto, premikanje po sobah . . .)

6.7 Rasterizacija

Tu se primitivi pretvorijo v diskretne elemente (fragmente) glede na del končne slike, ki ga pokrivajo (najmanj en fragment na piskel). Izvede se interpolacija lastnosti oglišč (in/out interpolirank) v posameznih fragmentih. Nad posameznim fragmentom se nato izvede senčilnik fragmentov.

Rasterska slika je 2D polje slikovnih elementov (pikslov) - **diskretno**. Koordinate predmetov so **zvezne**. Rasterizacija pomeni učinkovit izračun **pokritja pikslov**. Poznamo dva načina:

- **stara šola** (malo velikih trikotnikov) - izračunamo robove in zapolnimo notranjost
  - **Bresenhamov algoritem**: $y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1$
  - ***Scan-line* rasterizacija**: procesiramo poligone po vrsticah; v vsaki vrstici najdemo presek z robovi poligona in zapolnimo piksle med robovi
  - problemi: večja kompleksnost scen pomeni veliko majhnih poligonov
    - izračun robov je lahko počasen
    - potrebno rezanje, če imamo poligone izven zalsona
    - težka paralelizacija
- **nova šola** (več manjših trikotnikov) - za vsak piskel izračunamo, če se nahaja v notranjosti trikotnika in ga izrišemo
  - ***groba sila***: za vsak piksel izračunamo, ali je znotraj trikotnika
    - izračunamo za piksle znotraj očrtanega pravokotnika
    - rezanje je enostavno
    - interpolacija lastnosti oglišč je enostavna
  - da ne računamo preveč nepotrebnih pikslov, očrtan pravokotnik razbijemo na podpodročja

6.8 Senčilnik fragmentov

Svetloba pa shaderji pa tako.

6.9 Obdelava vzorcev

Tu se sestavi končno sliko:

- test škarij: odstrani fragmente, če so izven pravokotnika, ki ga lahko definiramo z *glScissor*
- test šablone: vsak fragment ima lahko **vrednost šablone**, ki se primerja s trenutno vrednostjo v **medpomnilniku šablone** (glede na definirano operacijo lahko fragment obdržimo ali zavržemo); uporaba za omejitev izrisa na področje, sence, mehke prehode, poudarjanje robov itn.
- test globine: določa vidnost (kateri poligoni bodo vidni)
  - vsak fragment ima globino (z vrednost v koordinatah zaslona)
  - vrednost se primerja z vrednostjo v **medpomnilniku globine** - fragment zavržemo, če je globina večja od globine v medpomnilniku globine
- mehčanje robov: piksle delimo na fragmente/vzorce, končno barvo piksla povprečimo
  - *Multisample antialiasing (MSAA)* (na robovih poligonov)
    - več vzorcev na piksel z majhnimi odmiki v koordinatah
    - globina in šablona se za vsakega ločeno izračunata
    - senčilnik fragmentov se izvede enkrat na piksel za vsak trikotnik
    - skupaj se vrednost povpreči preko vseh trikotnikov
  - *Supersample antialiasing (SSAA)*: senčilnik fragmentov se izvede za vsak vzorec
  - *Full scene antialiasing (FSAA)*: celotna scena se izriše na višji ločljivosti, slika zmanjša
- zlivanje: **zlivanje barve** fragmenta (S) z barvo slike (D) v rezultat (O); največ se uporablja za transparenco (**alfa** zlivanje)
- zapisovanje vrednosti: vrednosti se zapišejo v končne medpomnilnike - sliko (barva, alfa), globinski medpomnilnik (nova globina), medpomnilnik šablone (nova šablona)

6.9.1 Določanje vidnosti

Pri **slikarjevem algoritmu** izrišemo vse predmete od **zadnjega proti prvemu**. Imamo problem: kako urediti poligone. Ena rešitev je BSP drevo, vendar urejanje ni vedno mogoče - cikli.

Z buffer: za vsak fragment hranimo z vrednost trenutno najbližjega izrisanega poligona in izvedemo test globine - če je z vrednost manjša od vrednosti v medpomnilniku, fragment ohranimo, sicer ga zavržemo. Tu prekrivanje ni problematično. Z buffer hrani cela števila (pretvorimo z iz intervala $[0, 1]$ na $[0, 2^{d-1}]$). $1/z$ da večji poudarek na bližnje predmete, ker ni dobro, da je prednja ravnina preblizu očesu, saj bomo porabili preveč bitov za predstavitev predmetov zelo blizu le-teh. Lahko tudi pride do tega, da imata 2 predmeta enako globino, kar pripelje do utripanja pri izrisu.

Pri zgodnjem izločanju fragmentov lahko fragment zavržemo, čim vemo njegovo globino (to vemo že po rasterizaciji, pred senčilnikom fragmentov). S tem pohitrimo celotno zadevo, saj ga lahko preskočimo. Vendar je to smiselno samo, če izvedba senčilnika fragmentov ne vpliva na zakrivanje. Naletimo na težave, če imamo prosojnost ali če v senčilniku fragmentov spreminjamo globino oz. zavržemo fragment.

Problem Z buferja je, da hrani **le najbližji piksel**. A buffer, K buffer ... so variacije, ki hranijo seznam fragmentov, ki so aktivni v vsakem pikslu.

7. SENČILNIKI

7.1 "Efekti" s senčilniki

Tekstura ne spreminja barve, temveč vpliva na **izračun osvetlitve**. Vrednosti teksture definirajo **normale** na površino, s tem modeliramo hrapavo ploskev. Geometrije pri tem ne spreminjamo, rezultati so vidni zaradi modela osvetljevanja. Lepljenje normal dobro deluje, ko je predmet oddaljen; ko je blizu, opazimo, da predmet ni 3D, problemi tudi pri sencah.

V teksturi RGB komponente predstavljajo **XYZ normale**. Normale so lahko definirane v lokalnih koordinatah, če predmeta ne rotitamo. V splošnem se največ uporablja **tangentni** prostor. Za upodabljanje potrebujemo pretvorbo vektora luči in pogleda v ciljni prostor.

Tangentni prostor je definiran z:

- N: normala

- P_u: tangenta (vzporedna s smerjo teksture u)
- P_v: bitangenta oz. binormala (vzporedna s smerjo teksture v)
- $N = P_u \times P_v$

Implementacija lepljenja normal:

- tekstura v RGB vsebuje normalo v tangentnem prostoru
- skaliranje normal iz tekstur $[0, 255]$ na $[-1, 1]$
- pretvorba prostora: vektorja luči, pogleda v tangentni prostor ali normale v prostor pogleda
- izračun osvetlitve z novo normalo

Lepljenje izboklin je podobno kot lepljenje normal. Je starejša tehnika, ki je danes manj v rabi, ker je lepljenje normal bolj fleksibilno. Tekstura je navadno sivinska - višinska slika (predstavlja izbokline na predmetu). Za izračun izboklin vzamemo vrednosti v višinski teksturi $b(u, v)$. Točko P na predmetu bi navidezno spremenili v smeri normale: $P'(u, v) = P(u, v) + b(u, v)\vec{n}$. Razlika v višini nam pove, koliko moramo spremeniti normalo: $b_u = \frac{\partial b(u,v)}{\partial u} = b(u + 1, v) - b(u, v)$; $b_v = \frac{\partial b(u,v)}{\partial v} = b(u, v + 1) - b(u, v)$. Novo normalo izračunamo kot: $\vec{n}' = \vec{n} + b_v(P_u \times \vec{n}) + b_u(\vec{n} \times P_v)$.

Lepljenje odmika je podobno kot pri lepljenju izboklin, saj je tekstura sivinska slika. Tekstura **dejansko spremenja geometrijo**, torej položaj poligonov (model mora biti dovolj podroben, da tekstura dovolj dobro deluje). Poteka v **teselaciji** - GPU podpora za povečevanje ločljivosti poligonov in lepljenje odmika. Posledično lahko zmanjšamo kompleksnost modelov - detajle prepustimo teksturam.

Teselacija je **deljenje** črt in poligonov na manjše dele. Uporabljamo jo za nivoje podrobnosti (manjša podrobnost bolj oddaljenih modelov), pri lepljenju odmika in za risanje parametričnih krivulj. Poznamo dva senčilnika:

- *Tesselation Control Shader*: določi kolikokrat delimo poligon/črto
  - *inner*: število gnezdenih primitivov
  - *outer*: kolikokrat se deli vsak rob
- *Tesselation Evaluation Shader*: dobi položaj novega oglišča v težiščnih koordinatah; lahko interpolira med vrednostmi v ogliščih ali premika oglišča

Teselacijo torej lahko uporabimo tudi za implementacijo nivojev podrobnosti - bližnjim predmetom damo več podrobnosti s teselacijo in lepljenjem odmika, bolj oddaljene predmete pa izrišemo bolj grobo.

Senčilnik geometrije lahko ustvari novo geometrijo. Ideja: v programu generiramo točke, v senčilniku geometrije točko spremenimo v štirikotnik s teksturo. Implementacija:

- senčilnik oglišč: točko pretovrimo v prostor kamere
- geometrija: ustvarimo štirikotnik okoli točke in UV koordinate tekture
- fragmentacija: določimo barve

Želimo, da sceno obdaja **okolica**, ki jo podamo s teksturo. **CubeMap**: tekstura predstavlja vseh 6 smeri pogleda, nalepimo jo na kocko ki predstavlja okolico. Tu je poseben način lepljenja, kjer teksturo vzorčimo glede na smer proti kocki (vektor).

Z **lepljenjem okolice** lahko simuliramo **odboje svetlobe**. Smer odbite svetlobe določa teksle na teksturi. Barvo predmeta računamo kot **odboj** vektorja iz kamere na okolico.Upodabljanje: najprej izračunamo odboj vektora pogleda preko normale, nato pretvorimo v koordinate sveta in preberemo teksturo.

Sence povečajo realizem, saj nam dajo občutek globine. Imamo več možnosti za sence:

- vnaprej izračunane za statične predmete in zapečene v teksture (*lightmaps*)
- računanje med izrisovanjem (*shadow mapping*):
  - osnovna ideja: točka je **osvetljena**, če je **vidna** iz luči; vidnost iz luči izračunamo s **postavitvijo kamere** na položaj luči in izrisom sence

  1. prvi korak:
  - izris scene s postavitvijo kamere na položaj luči
  - shrani dobljeno globinsko sliko, ki ji rečemo **zemljevid senc**

  2. drugi korak:
  - izris scene s **položaja kamere**
  - na vsakem pikslu razdaljo do luči primerjamo z razdaljo shranjeno v zemljevidu senc (če je daljša smo v senci, če je enaka oz. krajša je piksel osvetljen)

- implementacijo drugega koraka izvedemo v senčilniku oglišč in senčilniku fragmentov

Pri **ambientnem zastiranju** so deli predmetov **zastrti** zaradi bližine drugih delov, posledično so zato temnejši. Ideja: *screen-space AO* - izračun **faktorja zastiranja** v vsakem fragmentu. V vsakem fragmentu vzorčimo okolico. Odstotek točk, ki je pod površino, je faktor zastiranja, z njim pa nato potemnimo sliko.

Pri **odloženem upodabljanju** izvedemo osvetlitev na zaslonskih pikslih. Imamo dva prehoda za izris:

- prvi: izračun podatkov za osvetljevanje (barva, globina, normale) v medpomnilnike
- drugi: uporaba shranjenih podatkov in izračun osvetlitve v prostoru slike

Prednost takega upodabljanja je **hitrost** - osvetljevanje je ločeno od geometrije. Osvetlitev izračunamo le za vidne piksle, če pa lučem omejimo še domet, lahko še zmanjšamo število izračunov - za vsako luč računamo osvetlitev le za piksle, ki jih osvetli. Naletimo tudi na določene težave: različni materiali zahtevajo več medpomnilnikov, težje mehčanje robov, ni transparence. Implementacija:

1. v medpomnilnike shranimo vsaj barvo, normale, globino/položaj
2. za vsak vidni piksel vemo globino, barvo in normalo

- izračunamo osvetlitev - končno barvo
- upoštevamo le luči, ki dosežejo piksel
- celo sceno lahko izrišemo

7.2 Parametrične predstavitve

To je neposredna predstavitev predmetov s krivuljami in ploskvami, ki jih krivulje definirajo. Veliko se uporabljajo v CAD aplikacijah in animacijah. Z njimi dosežemo gladkost. Poznamo dva načina predstavitev krivulj:

- eksplicitne: $y = f(x)$ - enostavno vemo, kdaj točka leži na krivulji
- implicitne: $f(x, y) = 0$ - funkcije z večkratnimi vrednostmi, enostavno vemo, kdaj točka leži na krivulji
- parametrične: $x(t) = f_x(t)$, $y(t) = f_y(t)$ - funkcije z večkratnimi vrednostmi - neodvisnost od k.s.

Splošen zapis parametrične krivulje: $p(t) = \sum_i p_i B_i(t)$.

- p_i: kontrolne točke
  - določajo obliko (in položaj) parametričnih krivulj
  - določajo točke ob/skozi nekatere naj bi krivulja potekala: $p_i = [x_i, y_i, z_i]$
- $B_i(t)$: mešalne funkcije (*blending functions*)
  - določa kakšen je vpliv točke p_i na krivuljo za podan t
  - vsaka točka ima lahko drugačno funkcijo

Mešalne funkcije določajo tip krivulje. Navadno jih izberemo tako, da omogočajo lokalen vpliv - kontrolna točka naj ima največji vpliv v svoji okolici. Mešalne funkcije naj bodo:

- enostavne za izračun
- zvezne in obstaja naj odvod (po možnosti drugi)
- lokalne - interval, na katerem so različne od nič, naj bo majhen
- omogočajo interpolacijo - kontrolne točke naj ležijo na krivulji

Vse našteto pa težko dosežemo, zato opustimo lokalnost in interpolacijo.

Največkrat so mešalne funkcije polinomi. To pomeni, da je krivulja tudi definirana kot polinom. Stopnja določa, kakšna je lahko oblika krivulje, število kontrolnih točk je *stopnja* + 1. Pri tem največkrat uporabljamo **kubične polinome**.

Mešalne funkcije so t.i. **Bernsteinovi polinomi** (Bezierove krivulje?). V splošnem je stopnja poljubna, največkrat pa uporabljamo kubične krivulje. Bernsteinovi polinomi stopnje n: $B_{i,n}(t) = \binom{n}{i} t^i (1-t)^(n-i)$. Lastnosti Bezierovih krivulj:

- prva in zadnja kontrolna točka sta na krivulji, notranje pa ne
- premik točke ne vpliva na tangento v zadnji
- krivulja v celoti leži znotraj konveksne ovojnice kontrolnih točk
- na ravnini nobena črta ne seka krivulje večkrat kor njenih kontrolnih črt
- je afino invariantna (enak rezultat dobimo s transformacijo kontrolnih točk ali transformacijo točk na krivulji)

Enačbo Bezierove krivulje lahko zapišemo v matrični obliki, ki omogoča učinkovit izračun krivulje glede na podani t na grafični strojni opremi. $p(t) = \sum_{i=0}^{p_i} B_{i,3}(t) = p_0(1-t)^3 + p_1 3t(1-t)^2 + p_2 t^2(1-t) + p_3 t^3$

Risanje:

- **enakomerno vzorčenje**: krivuljo izrišemo z ravnimi črtmi med točkami na krivulji
  - narišemo približek z N ravnimi segmenti (N izberemo vnaprej, izračunamo N točk na krivulji)
  - premalo točk nam da slab približek, preveč točk pa je počasn proces in segmenti se lahko prekrivajo
- **adaptivno vzorčenje**: število segmentov se prilagaja ukrivljenosti
  - malo segmentov, kjer je krivulja bolj ravna; veliko segmentov, kjer je ukrivljena
  - obstajajo različne sheme za določanje števila in lokacije segmentov

De Casteljaujevo deljenje je učinkovit algoritem za adaptivno vzorčenje. Vsak del kubične krivulje je tudi kubična krivulja (Bezierovo krivuljo lahko razdelimo v več manjših Bezierovih krivulj, te krivulje so bolj gladke kot celota). Krivuljo delimo, dokler ne dobimo skoraj ravnih segmentov, ki jih izrišemo kot črte. Kontrolne točke **rekurzivno razdelimo** na polovice - rezultat sta dve Bezierjevi krivulji, ki sta bolj ravni kot originalna. Ta postopek ponavljamo, dokler posamezna krivulja ni dovolj ravna, da jo izrišemo kot črto.

Za daljše krivulje navadno kombiniramo krivulje 3. stopnje - kombiniranim krivuljam pravimo **zlepki**. Zlepki naj bodo zvezni:

- zveznost C^0 se samo dotikajo
- zveznost C^1 - enak prvi odvod v stiku - enaka usmerjenost
- zveznost C^2 - enak drugi odvod v stiku - enaka ukrivljenost

Za Bezierove zlepke velja, da lepimo Bezierove krivulje 3. stopnje. Krivuljo razdelimo na N segmentov, vsak je parametriziran s t med $[0, 1]$ (končna točka je začetna naslednjega, rabimo $3N + 1$ kontrolnih točk). Zveznost C^1 lahko zagotovimo z dodatno omejitvijo: tangenta v stični točki obeh krivulj mora biti enaka. Taki zlepki imajo tudi slabosti: potrebujemo $3N + 1$ kontrolnih točk za N segmentov, težko dosežemo C^2 zveznost in del kontrolnih točk je na krivulji, del pa ne.

Uvedemo **B zlepke**, ki so posplošenje Bezierovih zlepkov. Kontrolne točke niso več na krivulji, za ekvivalenten zlepek potrebujemo manj kontrolnih točk kot pri Bezieru. Dosežemo lahko C^2 zveznost s kubilnimi mešalnimi funkcijami. Omogočajo lokalno kontrolo. Z njimi lahko ohranimo lastnost, da krivulja v celoti leži znotraj konveksne ovojnice kontrolnih točk. Ohranimo tudi afino invariantnost. Mešalne funkcije so rekurzivno definirane (ogabna formula).

...

NURBS so neenakomerni racionalni B zlepki. So najbolj standardne krivulje v 3D oblikovanju in predstavljajo posplošitev Bezierovih krivulj in B-zlepkov. Vozli navadno **niso enakomerno razdeljeni**. Vektor vozlov je tako v NURBS pomemben, saj določa bazne funkcije. Imamo lahko večkratne vozle. Večji skoki pomenijo večji (daljši) vpliv baznih funkcij.
Pri NURBS je C^2 zveznost možna s kubičnimi zlepki, ki so lokalni. Krivulja v celoti leži notraj konveksne ovojnice kontrolnih točk. Poleg afine invariatnostni zagotavljajo še projekcijsko invariantnost.

Parametrične ploskve so razrešitev krivulj v 2D. Namesto enega parametra $f(t)$ imamo dva - $f(u, v)$ in 2D množico kontrolnih točk. Ploskev "lepimo" iz posameznih parametričnih **krp**. Krpe so v obeh smereh sestavljene iz krivulj. Večja ploskev je sestavljena iz več krp, ki si delijo robne točke. Uporabljamo jih, kadar želimo natančen matematičen model površine. Problematične so za splošno modeliranje, saj je težavno sestavljanje različnih ploskev. V animaciji, filmih se zato bolj uporabljajo deljene ploskve ali kombinacija NURBS in deljenih ploskev.

8. BARVE

Bitna slika je 2D tabela **slikovnih elementov - pikslov**. Piksel je najmanjši element slike. Vsebuje zapis barve (glede na zapis barve imamo črno-bele, sivinske, barvne, večspektralne ... slike; glede na število bitov za zapis barve imamo 8, 16, 24, 32 ... bitno globino slike).

Vidna svetloba je elektromagnetno valovanje (390-750 mm). Oko zazna množico fotonov, ki nanj "padejo". V računalniški grafiki imamo poenostavljen pogled na barve: "žarki" svetlobe imajo neko smer in nosijo celoten spekter. Barvo vi lahko zapisali kot **celoten spekter** in ga reproducirali, kar pa bi bilo potratno. Poznamo **tridržljajsko**

teorijo: katerokoli barvo lahko predstavimo s kombinacijo **treh osnivnih barv** (lahko reproduciramo vse barvne odtenke, ki jih lahko zaznamo).

Barvni prostor je nabor parametrov, ki določajo, kako prikažemo poljubno barvo. Barvni prostor **sRGB** je najpogostejši RGB prostor. Uporabljajo ga monitorji, praktično vsi neprofesionalni fotoaparati, ... Poleg sRGB poznamo še AdobeRGB, PhotoPro, ...

Barvni prostor **CIE XYZ** uporabljamo za pretvarjanje med barvnimi prostori. Z njim lahko predstavimo **vse vidne barve**. Ta prostor je umetno ustvarjen (nima fizikalnega ekvivalenta). Za lažjo vizualizacijo XYZ prostora, so uvedli prostor xyY (Y je svetlost, xy sta deleža barvnih vrednosti - $x = \frac{X}{X+Y+Z}$, $y = \frac{Y}{X+Y+Z}$). Barvni diagram CIE je izrisan prostor xy - vidne barve so v podkvasti obliki; obodu rečemo krivulja spektralnih barv in predstavlja enobarvne svetlobe; mešanica dveh barv na diagramu leži na črti, ki povezuje barvi.

Barvni obseg vsake naprave, ki deluje na mešanju primarnih barv, je nabor vseh barv, ki jih naprava lahko reproducira. Velja, da z mešanjem osnovnih barv lahko ustvarimo le barve, ki ležijo znotraj **konveksne ovojnice** osnovnih barv na barvnem diagramu CIE. Točke izven konveksne ovojnice bi ustrezale negativnim vrednostim.

RGB barvni model temelji na seštevanju barv, za tiskalnike/črnilo pa potrebujemo drug barvni model, saj črnilo odšteva - **CMY**. Ker CMY je more dobro reproducirati temnih delov, dodamo še K - black $\Rightarrow$ **CMY(K)**. Barvni obseg naprave, ki uporablja CMYK barvni model, lahko prav tako predstavimo na barvnem diagramu CIE.

OS in aplikacije za konverzije med barvnimi prostori uporabljajo **sistem za upravljanje z barvami (Color Management System)**. Če želimo recimo natisniti sRGB sliko posneto z mobilnim fotoaparatom, jo CMS najprej pretvori iz sRGB v XYZ, nato pa tiskalnik iz XYZ v CMYK.

RGB barvni model ni intuitiven za določanje barv v risarskih programih. Alternativni predstavitvi za RGB model sta **HSL** in **HSB** (hue, saturation, lightness/brightness). Omogočata nam bolj intuitivno upravljanje z barvami. To **nista barvna prostora**, ampak le drug zapis barve v RGB modelu.

Za percepcijo svetlosti velja **Webrov** zakon: $\frac{\Delta I}{I} = k$. Posledično svetlosti oz. barv ni smiselno kodirati linearno in porabiti enako bitov za vse intenzitete. **Gama korekcija** pomanjkljivost odpravlja s tem, da sliko ob zajemu zakodiramo $(1/\gamma)$. Z gama korekcijo v računalnikih upravlja sistem za upravljanje z barvami. Barvni prostori lahko definirajo svoje načine gama kodiranja.

9. GLOBALNO OSVETLJEVANJE

Slika nastane kot interakcija med **objekti** v sceni (imajo položaj, material, ki določa kako interaktirajo s svetlobo), **lučmi** (imajo položaj, barvo svetlobe, obliko, smer širjenja svetlobe) in **gledalcem** (položaj gledalca oz. kamere določa kaj vidimo oz. kaj je na sliki). Poznamo lokalno in globalno osvetljevanje.

Pri globalnem osvetljevanju metode večinoma temeljijo na **sledenju žarkom**:

- sledenje žarku: osnovna metoda; zrcalni odboji, ni difuznih odbojev
- sledenje poti: stohastičen algoritem; difuzni in zrcalni odboji

9.1 Sledenje žarku - ray tracing

Je metoda globalnega osvetljevanja, ki simulira svetlobne žarke. V realnosti žarke generirajo svetlobni viri in potujejo do očesa. Za tovrsten izračun moramo slediti veliko žarkom, le malo pa jih pride do očesa. Pri metodi sledenja žarkom obrnemo situacijo - sledimo žarkom od očesa preko vseh pikslov v sliki in gledamo kam se zaletijo.

Metanje žarka je prvi del algoritma sledenja žarku. Koncept:

- sledimo žarku svetlobe od očesa do presečišča s prvim predmetom
- en žarek skozi vsak piskel v končni sliki
- v presečišču s predmetom izračunamo barvo z osvetlitvenim modelom
- če žarek ne seka nobenega predmeta, je piskel črn

Da za nek piksel slike izračunamo **smer žarka**, moramo **konstruirati žarek**:

- koordinatni sistem kamere je $[\vec{u}, \vec{v}, \vec{w}]$
- slika (velikosti $n_x \times n_y$) je pravokotna na $\vec{w}$ kamere in na radzalji d od kamere
- koordinate (xu, v, w) piskal (i, j) v koordinatah kamere so:

  - $u = l + (r-l)\frac{i+0.5}{n_x}$
  - $v = b + (t-b)\frac{j+0.5}{n_y}$
  - $w = -d$
- izvor žarka je e
- smer žarka: $\vec{d} = u\vec{u} + v\vec{v} - d\vec{w}$
- piksel na sliki: $p = e + \vec{d}$

Dobimo parametrično enačbo žarka: $r(t) = e + t(p-e) = e + t\vec{d}$

Da ugotovimo, ali in kje žarek preseka predmet moramo poiskati vrednost t v enačbi $r(t) = e + t\vec{d}$, pri kateri žarek preseka nek predmet. Najmanjši $t > 0$ bo najbližji presek. Iskanje presekov je časovno najbolj zahteven del pri tej metodi in iz nje izpeljanih.

Presek s kroglo: $(p-c)\cdot(p-c) - r^2 = 0 \Rightarrow (e+t\vec{d}-c)\cdot(e+t\vec{d}-c) - r^2$; 2 realni ničli - manjša je prvi presek; dvojna ničla - tangenta; ena pozitivna ena negativna ničla - žarek se začne v krogli in gre vn; kompleksni ničli: žarek ne seka krogle. Za izračun osvetlitve v točki preseka potrebujemo tudi **normalo**. Pri krogli je to enostavno: $\vec{n} = \frac{p-c}{|p-c|}$.

Presek s trikotnikom: v **težiščnem** k.s. predstavimo točko p kot $p = a + \beta(b-a) + \gamma(c-a)$. Nato na mesto p vstavimo enačbo žarka in dobimo sistem treh enačb in treh neznank. Če velja $t > 0$ in $0 < \gamma < 1$ in $0 < \beta 1 - \gamma$, je točka znotraj trikotnika.

Algoritmov za določanje presekov je cela vrsta, ker je to časovno najbolj kritičen del algoritma metanja žarka.

Če želimo najti prvi presek v sceni moramo najprej izračunati presek z vsemi predmeti v sceni. Vrnemo presek z najmanjšim t, ki je večji od 0.

Ko najdemo presek žarka s prvim telesom, izračunamo še **osvetlitev** (v vsaki točki/pikslu slike). Osvetlitveni model si lahko izberemo.

Metanje žarka upošteva tudi **sence**. V vsaki točki preseka pošljemo senčni žarek (*shadow ray*) proti vsaki luči. Če na poti senčnega žarka najdemo kak presek s predmetom, je točka v senci, sicer ni.

Žarku sledimo tudi po prvem dotiku s predmetom v dve smeri: **popolni**odboj za zrcalne odboje in **prepuščeni** žarek za prosojne materiale. Pri obeh novih žarkih rekurzivno ponovimo celoten postopek sledenja in seštevamo svetlobne prispevke žarkov. Rekurzijo pri nekem žarku ustavimo:

- ko žarek zadane luč (dobi bavro luči)
- ko žarek ne zadane ničesar (tema)
- ko žarek pride do predmeta, ki nima zrcalnih odbojev in ni prosojen
- omejiti moramo tudi globino rekurzije

Odbiti žarek izračunamo kot popolni odboj:

- $\vec{m}? - \vec{n}(\vec{n}\cdot\vec{d})$
- $\vec{s} = \vec{d} + \vec{m}$
- $\vec{r} = \vec{m} + \vec{s} = \vec{d} + 2\vec{m} = \vec{d} - 2\vec{n}(\vec{n}\cdot\vec{d})$

Nov odbiti žarek je torej parametrično zapisan kot $r(t) = p + t\vec{r}$.

Prosojni materiali prepuščajo svetlobo, s tem pride do **loma svetlobe**. Svetloba se upočasni, ko preide v bolj gost medij. Velja **Snellov zakon**: $\eta_d \sin\theta = \eta_t \sin\theta_t$ (η_d je lomni količnik v snovi vstopnega žarka, η_t je lomni količnih v snovi prepuščenega žarka); $\eta = \frac{\text{hitrost svetlobe v vakuumu}}{\text{hitrost svetlobe v mediju}}$. Pri vstopu v hitrejši medij lahko pride do **popolnega** notranjega odboja. Smer prepuščenega žarka izpeljemo iz Snellovega zakona in dobimo: $\vec{t} = \frac{\eta_d}{\eta_i}(\vec{d} - \vec{n}(\vec{n}\cdot\vec{d})) - \vec{n}\sqrt{1 - \frac{\eta_d^2}{\eta_i^2}(1-(\vec{n}\cdot\vec{d})^2)}$.

Osvetlitev v točki računamo kot pri metanju žarka, le da prištejemo še delež svetlobe I_r in I_t, ki pridejo od obeh novih žarkov: $I = k_a L_a + V_i L_i (k_d(\vec{L}\cdot\vec{N}) + k_s(\vec{V}\cdot\vec{R})^p) + k_r I_r + k_t I_t$.

Osnovni algoritem sledenja žarku ima pomanjkljivosti, izgled slik je precej umeten (ostri robovi, trde sence, vse je v fokusu, ...). Veliko pomanjkljivosti odpravlja t.i. **sledenje porazdeljenim žarkom**. Namesto enega žarka delamo v vsaki fazi z več žarki.

Pri **mehčanju robov** namesto enega žarka iz očesa pošljemo skozi vsak piksel več žarkov (navadno žarke naključno stresemo). Končna barva piksla je uteženo povprečje barv vseh žarkov, tako pa dobimo bolj mehke prehode med ostrimi kontrasti.

Točkasta luč ni realističen model, saj povzroča trde sence, zato uvedemo površinsko luč, ki zajame večjo površino. Da dobimo **mehke sence** v luč pošljemo več (nekoliko naključno porazdeljenih) **senčnih žarkov**, porazdeljenih po njeni površini in seštejemo doprinose.

Pri standardnem sledenju žarkov so odboji **preveč podobni zraclu**, zaradi grobosti materialov pa so v realnosti precej bolj zabrisani. Idejo prenesemo tudi na odbite žarke; namesto enega jih ustvarimo več v (nekoliko naključnih) smereh, s tem dobimo **mehki odboj**.

Prosojnost (difuzna prosojnost) računamo kot za odbite žake, lahko tudi pri prepuščenih žarkih ustvarimo več "naključno" **porazdeljenih žarkov** okoli idealnega žarka, s tem pa dobimo efelkt prosojnosti oz. polprozornosti.

Globinska ostrina je področje okoli fokusne razdalje, v katerem je slika še vedno sprejemljivo ostra. Pri RG kameri je ostrina idealna, vsi žarki gredo iz ene točke, vse je ostro. **Leča** je bolj realističen model, saj so žarki razporejeni po leči. Globinsko ostrino simuliramo s **porazdelitvijo žarkov po površini leče**. Slika je postavljena na fokusno ravnino.

Do **zabrisanega gibanja** pride, ko žarke porazdelimo po času in povprečimo. Pri tem bodo zamegljeni tisti predmeti, ki se premikajo.

Sledenje žarkov je **počasna metoda**, predvsem zaradi potrebe po računanju presekov. **Hierarhija očrtanih teles** je metoda, s katero predmete očrtamo s hierarhijo teles, s katerimi lahko enostavno izračunamo preseke - **očrtana telesa**. Ko sledimo žarkom, začnemo pri korenu drevesa in nadaljujemo proti predmetom v listih. Uporabimo lahko tudi druge tehnike razdelitve prostora. Hierarhično računamo preseke žarka, dokler ne pridemo do posameznih primitivov, s katerimi računamo presek.