|  |
| --- |
| **UNIVERSITATEA BUCUREșTI**  **facultatea de matematică - informatică**  **LUCRARE DE LICENță**  **-** |
| **Ray Tracing Distribuit în rețea** |
|  |
| **Profesor coordonator:**  Gheroghe Ștefănescu |
| **Student:**  Alexandru Gherghe  BUCUREȘTI |
| Iunie 2015 |

Continut:

[**1.** **Ray-Tracing – noțiuni introductive** 2](#_Toc420969771)

[1.1 Procesul Vizual 2](#_Toc420969772)

[1.2 Modelarea procesului vizual pe calculator 3](#_Toc420969773)

[**2.** **Elemente de geometrie în ray – tracing** 6](#_Toc420969774)

[2.1 Sfera 7](#_Toc420969775)

[2.1.1 Intersecția Rază – Sferă 7](#_Toc420969776)

[2.1.2 Găsirea normalelor la suprafeța sferei 10](#_Toc420969777)

[2.2 Planuri și suprafețe plate 11](#_Toc420969778)

[2.2.1 Intersecția rază – plan 12](#_Toc420969779)

[2.2.2 Intersecția rază – poligon 13](#_Toc420969780)

[2.2.3 Regula par-impar 13](#_Toc420969781)

[**3. Elemente de fizică în Ray Tracing** 16](#_Toc420969782)

[3.1 Conexiunea dintre lumină și culoare 16](#_Toc420969783)

[3.1.1 Pachete de undă 18](#_Toc420969784)

[3.1.2 Convertirea lungimii de unde în spectrul RGB 19](#_Toc420969785)

[3.1.3 Curbe de reflexie 20](#_Toc420969786)

[3.1.4 Implementarea distribuților spectrale și a coeficientului de reflexie 21](#_Toc420969787)

[3.2 Interacțiunea lumină - suprafață 22](#_Toc420969788)

[3.2.1 Iluminare și umbre 23](#_Toc420969789)

[3.3 Reflexia difuză 25](#_Toc420969790)

[3.4 Reflexia speculară 27](#_Toc420969791)

[3.5 Refractia luminii 32](#_Toc420969792)

1. **Ray-Tracing – noțiuni introductive**

O problemă generică în grafica pe calculator o reprezintă modalitatea de a reprezenta puncte din planul tridimensional în planul bidimensional astfel încât acestea să poată fi desenate pe ecran.

Procesul de a transforma date din spațiul tridimensional în spațiul bidimensional se numește randare tridimensională. De la vizualizarea de date complexe din punct de vedere grafic și până la crearea unor efecte vizuale de înaltă calitate, întâlnite atat de în jocuri, filme, pe televizoare, pe calculatoare, randarea tridimensională este nucleul modului de interacțiune al calculatoarelor cu ochiul uman și implicit cu oamenii.

Ray Tracing este o tehnică de randare tridimensională capabilă să producă imagini realiste de foarte înaltă calitate având ca pornind de la date geometrice aparținând unei scene. Pentru a înțelege cum funcționează această tehnică, este necesar să studiem și să înțelegem procesul creierului de a prelucra stimulul vizual.

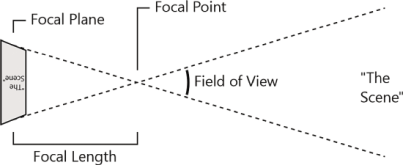
## 1.1 Procesul Vizual

Atunci când ne uităm la un obiect, există un șir lung de evenimente care au loc până la vizualizarea efectivă a respectivului obiect. Lumina este emisă dintr-o sursă aflată în apropierea obiectului. O parte din această lumină se îndreaptă către obiect, o altă parte circulă în alte direcții. Din lumina care se îndreaptă către obiect, o parte a ei efectiv întâlnește obiectul în cauză. O parte din lumină va fi deviată de pe acesta iar o altă parte va fi absorbită de ochiul uman.

Lumina ajunge în ochiul uman prin pupilă care are rolul de a fi punctul focal. De aici lumina este proiectată pe retină unde terminațiile nervoase trimit informația despre lumină către creier. Aceste semnale sunt interpretate de către creier și procesate în imaginea văzută.

Pentru a avea o imagine clară, avem nevoie de un punct focal. Dacă restricționăm lumina astfel încât să treacă prin punctul focal, ne asigurăm de faptul că toată lumina care ajunge în ochi a trebuit sa urmeze același traseu, deci a provenit din aceași locație. În caz contrar, lumina care ajunge în ochi ar fi putut proveni de oriunde, creând o imagine confuză, în ceață.

Camerele foto imită acest proces. Lentilelele lor au un punct focal care elimină lumina străină, în timp ce lumina care trece prin punctul de contact se proiectează pe un plan în partea din spate a camerei, similar cu retina în spatele ochilor noștri. Acest plan este foto-senzitiv, înregistrând tipul de lumină care ajunge la el. Folosind aceste date, putem reconstrui imaginea.

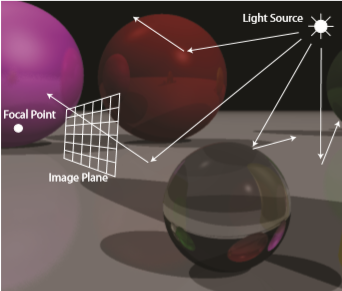


*Figura 1: Lumina din scenă traversează retina prin pupilă și proiectată pe retină în spatele ochiului*

## 1.2 Modelarea procesului vizual pe calculator

Acest proces poate fi simulat și pe calculator. Ne vom folosi de un punct arbitrar ales ca punct focal. De asemenea, ne va trebui un plan de proiecție care va fi amplasat în fața punctului focal, nu în spatele acestuia. Punctul focal nu trebuie să se afle neapărat în fața planului imaginii, vrem doar să ne asigurăm că lumina va trece prin el la un moment dat. Ne putem imagina acest plan ca un geam de sticlă prin care privim către scenă. Vom împărți acest plan în pixeli, urmând apoi să trimitem raze de lumină către scenă.

Atunci când lumina traversează scena înspre punctul focal, vom colora pixelii prin care lumina trece în mod corespunzător interacțiunii cu aceasta din urmă.

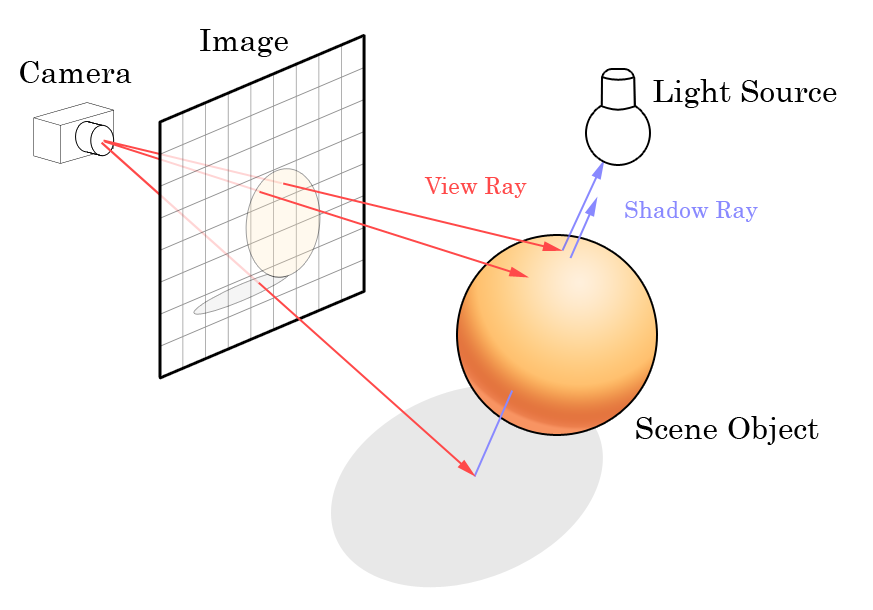


*Figura 2:*

Acest model simulează ceea ce se întâmplă în realitate însă în practică este ineficient de implementat. Asta deoarece șansele ca o raza de lumină aleatoare care străbate planul imaginii să ajungă în punctul focal sunt foarte mici. Nu avem puterea computațională de a modela suficienți de mulți fotoni pentru a ne asigura că măcar o parte din ei o să ajungă în planul imaginii.

**1.3 Ray Tracing**

Ray Tracing foloseste inversul acestui model! În loc să traseze raze de lumină de la sursa de lumină către planul imaginii, trasează raze de lumină dinspre planul imaginii către scenă.



*Figura 3: Raze de lumină sunt trimise către scenă prin planul imaginii.*

Aceste raze au rolul de a strânge informații suficiente pentru a colora corespuzător pixelii necesari din planul imaginii.

Când acestea întâlnesc un obiect, examinăm punctul de coliziune pentru a determina ce fel de lumină ar putea fi reflectată din acel punct și înapoi de-a lungul razei incidente către scenă. Procesul computațional se reduce drastic, din moment ce nu mai generăm raze de lumină care nu contribuie cu nimic la rezultatul final.

Algoritmul de ray – tracing poate fi succint prezentat în felul următor: pentru fiecare pixel din imagine, trasăm o rază din punctul focal prin pixelul respectiv al planului imaginii. Dacă raza intersectează vreun obiect din scenă, căutăm cel mai apropiat punct de intersecție. Noi putem vedea cel mai apropiat obiect de noi, orice se află în spatele acestuia fiind blocat. Se culeg informații despre lumina care ar putea fi reflectată în punctul de intersecție cu planul focal. Având aceste informații, calculăm culoarea finală din pixelul respectiv.

Ray Tracing are câteva avantaje față de alte tehnici de randare tridimensională. Datorită faptului că simulează foarte bine mecanismul de funcționare al ochiului uman, poate produce imagini de un realism foarte ridicat deoarece efectele de lumină ca reflexia, refracția, difuzia, umbrele sunt natural rezolvate de ray tracing datorită modului lui de funcționare.

Alte tehnici de randare tridimensională au nevoie de diferite extensii pentru se putea descurca cu aceste efecte de lumină.

Cu toate acestea, ray tracing-ul poate necesita multe calcule, facându-l încet la randarea de imagini de dimensiuni foarte mari, sau la randarea scenelor cu foarte multe obiecte. Datorită acurateții și realismului său, este folosit atunci când randarea se poate realiza dinainte, cum ar fi în filme. Deși nu este folosit în aplicații în timp real datorită puterii foarte mare de calcul care îi este necesara, acest lucru se poate schimba odată cu trecerea timpului, performanțele calculatoarelor crescând în mod constant.

Implementarea acestui algoritm necesită cunoștințe de geometrie și fizică. Pentru început, trebuie sa găsim punctele de intersecție între razele generate și obiectele din scenă. Aceste puncte trebuiesc ulterior analizate pentru a determina proprietățile luminii care se reflectă de pe ele înapoi în planul imaginii.

1. **Elemente de geometrie în ray – tracing**

Problema fundamentală al oricărui program ce implementează ray tracing constă în a găsi intersecțiile rază – obiect din scenă. Algoritmii folosiți pentru intersecțiile rază-obiect determina următoarele informații: cel mai apropiat punct de intersecție dintre rază și un obiect din scenă, cât și normal la suprafata obiectului în acel punct. Normala la suprafata obiectului va juca un rol important în fizica din spatele reflecției luminii.

Strategia de bază pentru a în algoritmii de intersecție este să folosim ecuația generală pentru respectivul obiect cât și parametrii care descriu raza și a rezolva sistemul de ecuații găsind punctul de intersecție (sau punctele, depinzând de suprafață).

O rază poate fi descrisă ca un vector linie cu o constrângere.

Unde p este punctul de pornire al razei iar v este vectorul direcție. Obligând λ să fie pozitiv, ne asigurăm de direcția razei în sensul corect. Împărțind vectorul de-a lungul componentelor, obținem parametrii ecuației pentru rază:

Combinând această ecuație parametrică cu ecuațiile definitorii ale diferitelor obiecte obținem o formulă generală pentru găsirea intresecțiilor cu un obiect.

## 2.1 Sfera

Sferele sunt un punct natural de plecare în programele de ray – tracing deoarece au ecuații simple și exemplifică precis la ce excelează algoritmii de ray – tracing și anume randarea suprafețelor curbate. Pentru început trebuie să determinăm dacă o rază data r intersectează sfera.

### 2.1.1 Intersecția Rază – Sferă

Ne gândim la sferă ca la o colecție de puncte care sunt echidistante față de un centru c și o rază r. Un punct q se află pe cerc dacă și numai dacă:

Deci un punct se află pe cerc dacă magnitudinea vectorului din centru către acel punct este r. Pentru a găsi intersecția razei cu cercul substituim q cu ecuația razei.

Deci acum putem rezolva ecuația pentru λ dat și ne vom folosi de rezultat în formula pentru rază pentru a găsi punctele de intersecție.

Formula pentru magnitudinea vectorului este:

Rearanjând factorii, obținem:

Care se simplifică în:

Reamintim că o ecuație de gradul 2 de forma:

are rădăcinile

***În cazul nostru:***

Din moment ce coeficientul b este par, ne putem folosi de ecuația:

Cu rădăcinile:

Avem trei posibilități de a găsi rădăcinile și implicit și ecuațiile. Ori

Și nu avem rădăcini reale deci nici puncte de intersecție între rază și sferă, ori

Și avem o singură rădăcină, deci raza este tangentă sferei cu

Sau cazul

Însemnând deci 2 rădăcini și deci 2 potențiale puncte de intersecție date de:

În cazul în care ecuația are una sau două rădăcini trebuie facute verificări adiționale pentru a ne asigura că aceste puncte de intersecție sunt valide. Trebuie să ne asigurăm că este pozitiv, altfel punctele de intersecție s-ar afla pe partea greșită a razei. In cazul in care avem 2 soluții pozitive, o alegem pe cea mai mică deoarece ne interesează cel mai apropriat punct de intersecție.

### 2.1.2 Găsirea normalelor la suprafeța sferei

Odată găsit un punct de intersecție cu sfera, determinarea normalei la suprafață este trivială. În orice punct din sferă, normala suprafeței este vectorul din centrul sferei către acel punct. Deci pentru punctul de intersecție p, normala la suprafață, N este dată de:

Însă formula ne arată doar una dintre cele 2 normale la suprafață. În orice punct, o suprafață tridimensională are 2 normale din moment ce vectori N si –N sunt amândoi perpendiculari. Formula de mai sus este pentru normala dinspre exteriorul sferei, care este normala ce ne interesează dacă vectorul incident își are originea în afara sferei. De multe ori este ceea ce ne dorim însă dacă raza își are originea înăuntrul sferei, vom vrea normala internă, nu pe cea externă. Există un test pentru a determina care normală este cea căutată.

Fiind dat un vector I si vectorul normal N, considerăm unghiul dintre cei 2 vectori. Întotdeauna vom vrea ca normala să indice înspre direcția vectorului incident, deci ne dorim ca unghiul dintre cei 2 vectori să fie obtuz.

Putem folosi produsul scalar dintre 2 vectori pentru a testa. Produsul scalar a 2 vectori tridimensionali este dat de formula:

De asemenea, avem următoarea proprietate legată de unghiul dintre 2 vectori și produsul lor scalar :

Așadar, dacă , cosinus-ul unghiului este negativ (din moment ce magnitudinile trebuie sa fie pozitive) iar unghiul este obtuz. În acest caz ne dorim să avem N ca vectorul normal al suprafeței.

Dacă , unghiul este ascuțit iar vectorul incident își are originea în interiorul sferei. În acest caz returnăm –N ca vectorul normal al suprafeței.

De remarcat este faptul că dacă , vectorul incident este perpendicular pe normala suprafeței, deci raza incidentă este tangentă suprafeței.

În cazul sferelor, deducem că raza își are originea în afara sferei și deci putem returna N ca vectorul normal al suprafeței.

## 2.2 Planuri și suprafețe plate

În continuare vom studia intersecțiile rază – plan iar apoi ne vom extinde către poligoane mărginite.

### 2.2.1 Intersecția rază – plan

Un plan poate fi interpretat folosind vectorul normalei N și un punct din plan q.

Un alt punct p aparține planului dacă și numai dacă:

N (p − q) = 0

Știind că relația anterioară este intuitivă, deoarece dacă produsul scalar dintre N și este 0, atunci cosinus-ul unghiului dintre cei 2 vectori va fi 0, deci cei 2 vectori sunt perpendiculari. Din moment ce vectorul este perpendicular pe N, este paralel cu planul iar din moment ce un capăt al vectorului este prezent în plan, la fel va fi și celălalt capăt.

Deci, pentru un plan dat definit de N și q putem să substituim ecuația parametrică a razei în plan în locul lui p și să rezolvăm pentru pentru a găsi eventualele intersecții.

Extinzând produsul scalar, obținem

Distribuind componentele vectorului normal N, obținem o sumă de 2 produse scalare

Rearanjăm pentru a obține formula finală pentru

La fel ca la interscția rază – sferă, câteva scenarii sunt posibile. Dacă știm că vectorul normal și vectorul direcție sunt perpendiculari deci raza este paralelă cu planul. În acest caz, dacă și rezultatul coeficientului este nedefinit raza este paralelă cu planul dar nu în el, deci nu avem intersecție.

Dacă atunci vectorul este perpendicular pe plan deci raza se află în plan.

Dacă numitorul este ne-negativ și , atunci am găsit punctul de intersecție.

### 2.2.2 Intersecția rază – poligon

În geometria computațională, se pune problema de a afla dacă un anumit punct dintr-un plan se află în afara, înantrul sau pe marginea unui poligon. Este un caz special al găsirii locațiilor și are aplicabilități în domenii care se folosesc de procesarea datelor geometrice, cum ar fi grafica pe calculator, sisteme de informații geografice.

O descriere a acestei probleme în grafica pe calculator prezentă din 1973 arată două posibile abordări, trasarea de raze și însumarea unghiurilor.

### 2.2.3 Regula par-impar

O metodă simpla de a verifica dacă un punct este înăuntrul sau în afara oligonului este de a testa de câte ori o rază care pornește dintr-un anumit punct într-o direcție fixată întâlnește marginile poligonului.

Dacă punctul dat nu se află pe marginile poligonului, numărul de intersecții este un număr par dacă punctul se situează în afara poligonului sau un număr impar dacă punctul se situează înăuntrul poligonului.

Algoritmul este cunoscut și sub numele de”Regula par-impar”.

Acesta se bazeaza pe simpla observație că dacă un punct se mișcă de-a lungul razei de la infinit către punctul de probă și dacă intersectează marginile poligonului, posibil de câteva ori, atunci el se mișcă alternativ din exterior spre interior, apoi din interior spre exterior etc.

Ca un rezultat, dupa fiecare 2 intersecții cu marginile poligonului, punctul iese în exterior. Această observație poate fi demonstrată matematic folosind teorema lui Jordan.

Dacă este implementat pe un calculator cu precizie aritmetică finită, rezultatele pot fi incorecte dacă punctul se situează foarte aproape de marginile poligonului, datorită erorilor de rotunjire. În general, aceasta nu este o problemă deoarece viteza este mult mai importantă decât acuratețea perfactă, în majoritatea aplicațiilor din grafica pe calculator.

Totuși, pentru un program complet, se recomandă introducerea unei toleranțe numerice iar pentru un punct P care se află în această limită de toleranță, algoritmul ar trebui să raporteze faptul că punctul în cauză P se află foarte aproape de marginea poligonului.

Majoritatea implementărilor ale algoritmului de ray – casting verifică în mod consercutiv o rază cu toate fețele poligonului. În acest caz, se distinge o eventuală problema:

Dacă raza trece exact printr-un punct de intersecție a două fețe ale poligonului atunci raza va intersecta două segmente în capete. În acest caz, pentru a nu avea erori, ar trebui să numărăm doar o singură intersecție în acest caz, și nu două.

O problemă similară ar putea apărea pentru segmentele orizontale care ar cădea pe rază.

Problema este rezolvată în felul următor:

Dacă punctul de intersecție este un punct de pe o față a poligonului deja testată, atunci intersecția se pune la numărătoare decât dacă al 2- lea capăt al segmentului se află sub rază.

Acest lucru este echivalent cu a considera vârfurile pe rază ca fiind un pic deasupra acesteia.

Revenind, cazul unei raze care trece printr-un vârf al poligonului poate cauza probleme numerice acolo unde precizia aritmetică este limitată: pentru 2 părți adiacente aceluiași vârf, calcularea intersecției cu o rază s-ar putea să nu găsească vârful în ambele cazuri.

Dacă poligonul este dat prin mulțimea sa de vârfuri, atunci problema este eliminată verificând coordonata y a razei și capetele fețelor poligonului deja testate înainte de a calcula intersecția.

În alte cazuri, când fețele poligonului sunt calculate din alte tipuri de date, alte soluții trebuie aplicate pentru a se păstra robustețea numerică a algoritmului.

# 3. Elemente de fizică în Ray Tracing

O importanță mare în ray tracing o are determinarea intersecțiilor rază – obiect. Dar odată razele trimise către scenă, trebuie să cercetăm modul în care obiectele cu care se ciocnesc cât și lumina incidentă a acelor obiecte pentru a determina gradul de importanță a razei respective la imaginea finală.

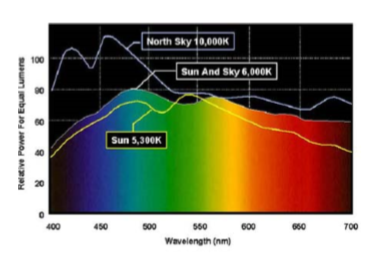
Câteva întrebări care survin:

* Obiectul este luminat sau se află în umbră?
* Dacă obiectul este luminat de o sursă de lumină, câtă lumină reflectă obiectul și cum schimbă reflexia lungimea de undă a luminii?
* Este obiectul capabil să reflecte lumina? Dacă da, se pot vedea obiecte reflectate pe suprafața lui? Cât de puternică este reflexia?
* Este obiectul translucid ? Cum modifică obiectul lumina transmisă ?
* Cum percepem un punct pe acest obiect? Ce culoare și intensitate a luminii vedem?

Vom examina ce trasee poate avea lumina atunci când se ciocnește de un obiect.

## 3.1 Conexiunea dintre lumină și culoare

Înainte de a discuta de interacțiunea dintre lumină și suprafață, trebuie sa înțelegem niște proprietăți de bază ale luminii. Lumina vizibilă este radiația electromagnetică a cărei lungimi de undă se află între 380 si 750 nanometri. Cand percepem lumina, foto-receptorii din ochii noștri nu absorb fotonii unul cate unul, ei absorb grupuri de fotoni pe care trebuie să îi interpreteze. In ray tracing, intalnim aceași problemă.



*Figura 4: Distribuția spectrală a soarelui [7]*

O sa modelăm acest grup ca un pachet de unde care contine grupuri de fotoni cu diferite lungimi de unde si intensități. Vom prezenta cateva example ale curbei spectrale în spectrul vizibil. De exemplu lumina solara are o curbă centratată pe spectrul vizibil.

Distribuția spectrală a unui bec incadescent demonstratează aceasta proprietate. Deoarece becurile degajă mai puțină caldură decat soarele, curba este orientată spre spectrul infraroșu.

Lămpile fosforescente au o distribuție oarecum diferită. Ele se bazează pe radiația produsă de atomi fosforescenți. Fiecare atom emite o bandă de radiație care produce aceste vârfuri in curbă, în loc de o curbă netedă.

Aceste example demonstrează varietatatea pachetelor de lumină. Deși formele sunt diferite, toate aceste example reprezinată lumină albă.

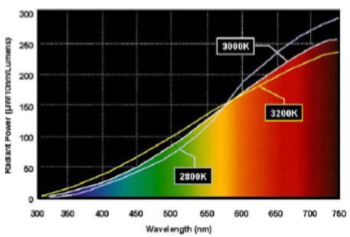
O întrebare care survine este de ce să reprezentăm aceste forme în algoritm dacă nu reprezentăm decât lumină albă.

Dorim să reprezentăm cât mai multă informație despre cadru pentru ca modelul să aproximeze realitatea cât mai exact.

Fiecare dintre examplele prezentate mai sus iluminează cadrul diferit datorită distrubuției lor. Lumina incadescentă va pune accentul pe obiectele roșii și galbene, în timp ce lumina fosforescentă va scoate în evidență nuanțele de albastru.

### 3.1.1 Pachete de undă

Pachetele de undă reprezintă o modalitate de a descrie un grup de fotoni. Pachetul este modelat sub forma unei curbe de-a lungul spectrului vizibil, curba reprezentând intensitatea energiei fotonilor la respectiva lungime de undă. Distribuția spectrală a soarelui este sub incidența legii corpului absolut negru.



*Figura 5: Distribuția spectrală a unui bulb incandescent de lumină [7]*

O altă distribuție spectrală guvernată de legea corpului absolut negru este cea a unui bulb de lumină incandescent. Însă, bulbii de lumină nu emit la fel de multă căldură coa soarele, deci sunt mai reci iar ca urmare curba este centrată în zona spectrului de infraroșu

### 3.1.2 Convertirea lungimii de unde în spectrul RGB

Transformarea lungimilor de undă în spectrul RGB poate fi un proces dificil. Pentru lungimile de undă 700nm, 546nm, 436 nm este usor, ele corespunzând culorilor primare: roșu, verde și albastru.

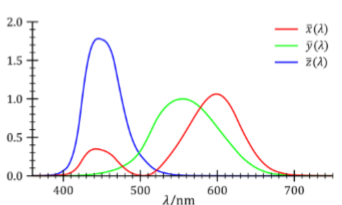
Cu toate acestea, multe culori si nuanțe nu au asociate o lungime de undă, ele fiind construite de ochiul nostru folosindu-se de o distribuție a lungimilor de undă.

De exemplu, nu există o lungime de undă pe care ochiul nostru sa o perceapă ca magenta, ea provenind dintr-o distribuție de roșu și albastru.

Avem nevoie de un algoritm care sa simuleze media pe care ochiul nostru o face pentru a transforma pachetele de lungimi de undă în culori.

Această problemă a fost studiată încă din 1930 de catre ICI și a rezultat in niște funcții care transformă lungimile de undă în culori din spectrul RGB.

S-a folosit un spațiu de culoare numit CIE XYZ, unde X, Y, Z sunt corelate cu valorile RGB standard. Odată obținute valorile XYZ, putem procesa în continuare pentru a obține valorile RGB standard.



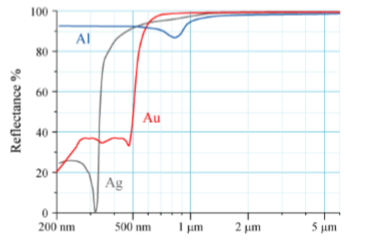
*Figura 6: Funcțiile de matching pentru culoare CIE X Y Z [1]*

Pentru a transforma o lungime de undă in coordonate XYZ o sa înmulțim funcția lungimii de undă cu funcțiile date pentru a obține valorile lui X, Y respectiv Z.

Ecuațiile sunt date de:

Odată ce am obținut culorea in spectrul XYZ, o vom traduce in spectrul RGB pentru a o afișa. Pachetul standard pentru acest spectru e sRGB care a fost creat de Hewlett-Packard și Microsoft. Transformarea este dată de urmatoarea ecuație:

### 3.1.3 Curbe de reflexie

****

*Figura 7: Curbe de reflexie pentru suprafețe metalice întalnite des [8]*

În majoritatea cadrelor, lumina nu este emanată de la sursă și ajunge la observator. Mai întâi ea se reflectă din obiectele aflate in scenă.

Când o rază de lumină lovește un obiect, unele lungimi de undă sunt absorbite, iar altele sunt reflectate înapoi in cadru.

Deci distribuția lungimilor de undă se schimbă in funcție de care lungimi de undă se reflectă și care sunt absorbite.

Vom modela acest proces prin a defini un coeficient de reflecție pentru fiecare obiect în parte. Coeficientul de reflecție este similar cu curbele spectrale care au fost definite pentru pachetele de unde. Acest coeficient ia valori intre 0 si 1, depinzând de cât reflectă dintr-o undă cu o lungime de undă dată.

Dacă un pachet de unde are distributia I și se reflectă dintr-o suprafață cu distribuția R, lumina reflectata va avea distribuția D dată de

)

### 3.1.4 Implementarea distribuților spectrale și a coeficientului de reflexie

Implementarea unui algoritm pentru procesarea și stocarea coeficienților de reflexie pentru diferite distrubuții ar fi foarte costisitoare. De aceea, vom reprezenta acești coeficienți ca și vectori evaluați la intervale regulate. În acest fel, vom reduce drastic timpul operațiilor pentru distribuții de lumină.

Dacă dorim să scalăm distribuția cu o constantă, trebuie doar să multiplicăm vectorul. Produsul a două distribuții este dat de produsul dintre componentele corespunzătoare. Integrala este obținută însumând componentele vectorului.

Putem să folosim un eșantion mai mare pentru a avea o aproximare mai bună sau putem folosi un eșantion mai mic pentru a crește viteza de procesare.

## 3.2 Interacțiunea lumină - suprafață

Atunci când lumina intersectează un obiect mai multe scenarii sunt posibile:

* Dacă suprafața este lucioasă asemenea unei oglinzi, lumina va fi reflectată de pe acea suprafață.
* Dacă suprafața este mată, ca un zid sau o foaie de hârtie, lumina se va împrăștia în mod egal în toate direcțiile.
* Dacă suprafața este transparentă de exemplu o bucată de sticlă, sau un diamant, lumina va trece prin ea, fiind refractată sub un unghi specific materialului prin care trece.

Acestea sunt exemple de reflecție speculară, reflecție difuză și refracție, 3 mecanisme de bază a interacțiunii lumină-suprafață.

În general, o suprafață nu este în totalitate speculară, difuză sau refractivă ci o combinație dintre acestea. Când o rază trimisă spre scenă intersectează un obiect, va trebui să luăm în calcul toate aceste lucruri pentru a calcula lumina pe care noi o vedem datorită acelei raze.

Dacă un obiect este luminat de o sursă de iluminare, trebuie să luam în considerare lumina care este reflectată difuz din acea sursă de lumină.

* Dacă suprafața este lucioasă, vom reflecta raza incidentă și vom trasa noua rază în scenă pentru a determina cum este lumina reflectată de acea suprafață.
* Dacă suprafața este transparentă va trebui să ducem raza refractată din obiect pentru a determina cantitatea de lumină transmisă mai departe.

Toate acestea determină cantiatea de lumină totală transmisa de raza incidentă.

Așadar, vom considera că orice suprafață va avea 3 coeficienți specifici pentru fiecare dintre mecanismele de transport a luminii de mai sus.

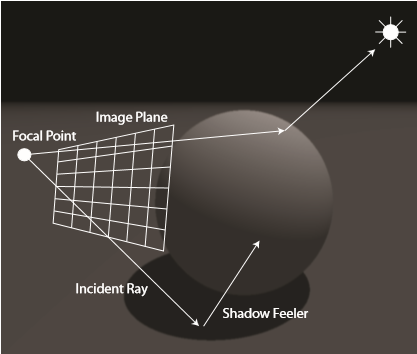
În general, când lumina întâlnește un obiect, ea va trebui analizată și etichetată făcând parte dintr-una din cele trei categorii menționate, astfel încât suma coeficienților să fie egală cu 1. Fiecare coeficient indică proporția de lumină care reflectată, reflectată difuz sau refractată.

### 3.2.1 Iluminare și umbre

Pentru a determina dacă un obiect, sau o parte din obiect apare în scenă, va trebui să aflăm mai întâi dacă obiectul în cauză este luminat de către sursele de lumină prezente în scenă.

În continuare vom prezenta algoritmul necesar pentru a realiza acest lucru.

* Dacă raza **r** ciocnește obiectul **o** în punctul **p**, începem să parcurgem toate sursele de lumină din scenă.
* Dacă sursa de lumină **i** se află în punctul , ducem o nouă rază din punctul p pe direcția vectorului . Această rază traversează distanța de la punctul de intersecție către sursa de lumină. Ducând această rază în scenă și verificăm dacă ea intersecteză vreun obiect până să ajungă la sursă.
* Dacă intersectează, atunci vom ști că punctul **p** nu este luminat de sursa de lumină **i**.
* Dacă raza nu intersectează niciun obiect până la sursa de lumină, punctul **p** este luminat de sursa de lumină **i**.



*Figura 8: Rază shadow feeler*

Aceste raze poartă denumirea de ”shadow feeler”. Putem folosi aceleași metode pentru a trasa asemenea raze în scenă, însă putem îmbunătăți performanța algoritmului oprind procesul în momentul în care am găsit un punct de intersecție mai apropiat de sursa de lumină deoarece este clar ca punctul se află în umbră din momentul în care am găsit un obiect aflat între el și sursa de lumină.

Folosind această abordare care se folosește de conceptul de ray-tracing, implementarea umbrelor este mult mai ușoara decât folosirea altor tehnici de rasterizare care necesită metode avansate pentru a randa umbrele în scenă.

De exemplu, o metodă de randare a umbrelor folosind OpenGL este de a construi un ray-tracer simplist care să suporte doar trasarea de raze de tipul ”shadow feeler”, necesare pentru a determina ce umbre ar trebui sa existe in scenă.

În cazul unor asemenea implementări, pot apărea erori de precizie aritmetică în timpul generării razelor dintr-un punct de intersecție dat. În acest caz, raza creată ar putea sa își aibe originea ușor în interiorul obiectului iar apoi să se lovească de acesta în punctul de intersecție sau foarte aproape de el.

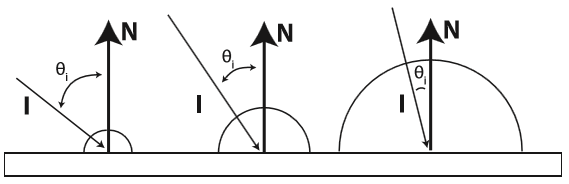
Ca și efect ulterior, părți din obiect vor fi umbrite de ele însăși. Pentru a evita acest scenariu, vom cere ca raza să parcurgă o anumită distanță înainte de a începe căutarea de intersecții cu alte obiecte.

## 3.3 Reflexia difuză

Reflexia difuză modelează împrăștierea care apare atunci când lumina lovește o suprafață aspră sau mată și reprezintă culoarea de bază a unui obiect. În timpul reflexiei, lumina și materialul obiectului interacționează foarte mult iar obiectul absoarbe sau reflectă lumina pe în funcție de curba lui de reflecție.

Într-o reflexie difuză ideală, lumina este împrăștiată în toate direcțiile în mod egal. Magnitudinea luminii reflectate este proporțională cu unghiul sub care lumina lovește suprafața și este independenta de unghiul de vizualizare al vectorului relativ la suprafața în cauză.

În practică, magnitudinea luminii reflectate este direct proporționala cu cosinusul unghiului dintre normala suprafeței și direcția luminii incidente, o proprietate cunoscută sub numele de ”legea cosinusului”.



*Figura 9: Magnitudinea luminii împrăștiate este proporțională cu*

Magnitudinea luminii reflectate va fi de asemenea proporțional cu cantitatea de lumină care este difuz reflectată, în contrast cu împrăștiat reflectată.

Așadar, modelarea reflecției difuze devine mai ușoară, având nevoie doar de vectorul normal al suprafeței și de vectorul în care lumina atinge suprafața, cât și de coeficientul de difuzie al suprafeței.

Când calculăm relecția difuză pentru un punct de la o sursă de lumină dată, va trebui întâi să determinăm dacă acel punct este luminat de sursa de lumină. Cunoscând deja modalitatea de a calcula daca un punct se află în umbră sau nu, putem calcula astfel vectorul direcție al luminii incidente din sursa de lumină în punctul dat, dacă obiectul nu este umbrit.

Dacă consideram punctul P de pe o suprafață cu coeficientul de difuzie d, vectorul normal al suprafeței N, curba de recflecție F(λ) cu lumină incidentă cu distribuția spectrului I(λ) venind de la vectorul L, vom calcula distribuția luminii reflectate difuz astfel:

Pentru a determina totalul reflecției difuze într-un punct dat, va trebui să parcurgem toate sursele de lumină existente care luminează respectivul punct și să însumăm rezultatele date de formula anterioară, pentru fiecare sursă de lumină în parte. În acest fel luăm în considerare doar reflexia difuză ce își are originea în sursele de lumină, pe când în realitate razele de lumină care nu își au originea în scenă pot de asemenea contribui la cantitatea totală de reflecție difuză.

Vom adăuga un pas în plus pentru a lua în calcul și lumina ambientală.

Pentru a calcula cantitatea totală de reflecție difuză într-un punct în care este prezentă și lumina ambientală, ne vom folosi de următoarea formulă

Metoda de mai sus este cea mai simplă metodă de a lua în calcul lumina ambientală. Totuși, există si alte abordări cu un grad de precizie mai ridicat.

O variantă ar fi să generăm aleatoriu raze de lumină din punctul respectiv către scenă pentru a colecta informații despre lumina prezentă din alte surse decât cele existente în scenă.

Această metodă ar oferi o precizie mai ridicată a cantității de lumină ambientală ce este prezentă într-un anumit punct, însă costul computațional devine ridicat.

## 3.4 Reflexia speculară

Reflexia speculară poate contribui în două moduri la lumina transmisă de un vector incident. O variantă este reflexia pur speculară, când lumina de oriunde din scenă se reflectă de pe suprafață pe planul imaginii.

Un exemplu de reflexie pur speculară este dat de modul în care lumina se reflectă din oglinzi sau o suprafață netedă cu indicie de relexie ridicat (apa dintr-un lac).

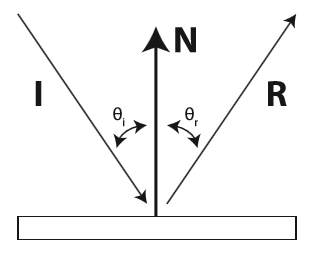
Suprafețele speculare pot de asemenea să reflecte lumina incidentă de la sursele de iluminare producând pete speculare luminoase.

Deoarece noi modelăm sursele de iluminare ca fiind puncte, algoritmul nu va lua în considerare lumina provenită direct din sursele de iluminare ca o posibilă sursă pentru reflexia pur speculară. De aceea, ne vom ocupa de petele speculare luminoase separat.

*Reflexie pur speculară*

Pentru a modela reflexia speculară, reflectăm raza incidentă din obiect și aplicăm algoritmul de ray tracing pe raza reflectată pentru a găsi lumină care are putea ajunge în obiect în punctul de intersecție.

Ulterior ne vom folosi de coeficientul specular al suprafeței în cauză pentru a calcula cantitatea reflexie speculară a razei de întoarcere.



*Figura 10: reflexie speculară,*

Întâi va trebui să găsim raza reflectată pentru a o trasa înspre scenă. Pentru a găsi vectorul de reflexie R al vectorului incident normalizat I de pe suprafața cu vectorul normal al suprafeței N, ne vom baza pe proprietatea că vectorul reflectat se va afla în același plan ca și vectorul incident și vectorul normal al suprafeței și că va forma același unghi cu vectorul normal al suprafeței ca și vectorul incident.

Din moment ce R, N, I se află în același plan, putem scrie R ca o combinație liniară a celorlalte două:

Dacă vom nega I, știm că N și –I formează același unghi format de N și R. Va trebui să negăm I astfel încât ambii vectori să aibă direcția opusă punctului de intersecție. Acest lucru implică:

Înlocuind R în formulă, obținem

din moment ce . Dacă luăm , obținem

Deci ecuația finală pentru vectorul reflectat este:

Combinăm vectorul R găsit cu punctul de intersecție pentru a găsi raza reflectată. Ducând această rază în scenă obtinem lumina care vine înapoi de-a lungul razei și se reflectă pe suprafață și în planul imaginii. Înmulțim R cu coeficientul specular al obiectului pentru a vedea cât de multă lumină reflectată contribuie la lumina finală returnată de raza incidentă.

În general o rază de lumină reflectată de pe o suprafață interacționează foarte puțin cu suprafața în cauză, așa că spectrul razei reflectate este aproape neschimbat de raza incidentă. Din acest motiv putem renunța la orice informația despre reflectanța suprafeței pentru reflexia speculară.

În lumea inconjuratoare există ceva interacțiune între suprafață și lumina incidentă, chiar dacă lumina este reflectată specular.

Curba de reflexie a unei suprafețe se va schimba proporțional cu unghiul luminii incidente. Pe măsura ce lumina incidentă se apropie de un unghi drept cu normala suprafeței, este mai puțină interacțiune între suprafață și lumină, ducând la o curbă de reflexie uniforma.

*Pete luminoase*

Petele luminoase apar atunci când lumina dintr-o sursă se reflectă dintr-o suprafata către observator.

Deoarece, sursele de lumina sunt considerate puncte, algoritmul nostru nu va găsi un punct de intersecție.

Pentru a trece peste acest impas, vom considera raza reflectată de raza incidentă și o vom compara cu vectorul vedere al observatorului.

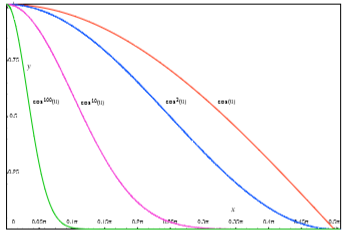
* Dacă unghiul este mic, raza se reflectă către observator și deci vom pune accent pe acea parte a obiectului.
* Dacă unghiul este mare, lumina este departe de observator deci nu vom pune accent pe obiect.

Suprafețe diferite dau naștere la diferite tipuri de pete:

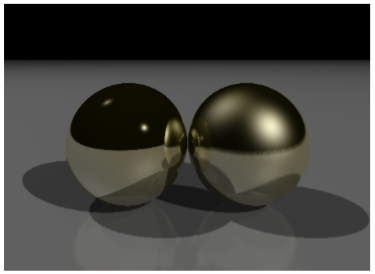
* O suprafață netedă va avea câteva pete foarte strălucitoare.
* În schimb, o suprafață dură va avea pete mai mari și distribuite uniform.

Pentru a simula acest effect, adaugăm un coeficient care modelează cât de netedă este o suprafață.

Pentru o sursă de lumină cu lungime de undă dată care întâlnește o suprafață cu coeficient specular **s** și coeficient de netezire **n** care se reflecta de-a lungul vectorului , privit de-a lungul vectorului **V**, avem ecuația:



*Figura 11: Curbele pentru cu exponent variabil*



*Figura 12: Nivel diferit de netezire a suprafeței metalului pentru a simula metal polisat*

Prezentăm o listă cu indici de refacție pentru cele mai întâlnite materiale.

* Aer 1.0003
* Apă 1.33
* Gheață 1.31
* Sticlă 1.485 - 1.926
* Diamant 2.419

Mai intâi negăm vectorul care ne arată din ce punct privim ca să fie in aceeași direcție ca și lumina reflectată.

Apoi aflăm cosinusul unghiului dintre cei doi vectori folosind produsul scalar a celor doi vectori.

Rezultatul este apoi ridicat la puterea dată de coeficientul de netezime. Dacă o suprafață este netedă, puține puncte vor fi scoase in evidență, evidențierea lor fiind mai puternică.

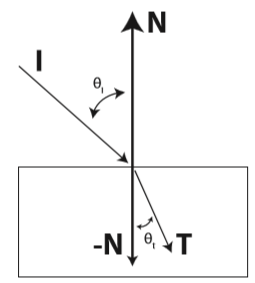
## 3.5 Refracția luminii

Pentru a modela refracția luminii trebuie să trasăm o nouă rază și să o urmărim prin cadru. Această rază trece prin interiorul obiectului.

Când lumina întâlnește un material, ea se refractă, deci își schimbă direcția. Putem observa acest efect atunci când privim o sticlă curbată sau un obiect parțial cufundat în apă. Acest lucru se obține datorită coeficientului de refracție al obiectului.

Indexul de refracție a unui material măsoară viteza lumii prin acel material. Dacă un material are indicele de refracție , lumina se mișcă de mai repede prin acel medium decât în vid.

Vidul are indexul de refractie 1. Indexul de refracție a diferitelor material poate fi găsit mai jos.



*Figura 13: și sunt legate de coeficienții de refracțile ale celor două medii.*

Când lumina întâlnește un material nou, ea se difractă datorită schimbării vitezei. Unghiul de difracție este obținut din coeficienți de difracție a materialul nou și materialul precedent.

Pentru o rază incidentă cu vectorul **I** care întâlnește o suprafață cu coeficientul de difracție care provine dintr-o suprafața cu coeficient de difracție într-un punct la care normală la suprafața este N, avem următoare relație între unghiul de incidență și unghiul de refracție:

Ca și în cazul reflexiei speculare, vom pune condiția ca vectorul refractat, vectorul incident și vectorul normal la suprafață să fie coplanar, deci vectorul refractat se poate scrie ca o combinație liniară de forma:

Acum putem gasi formula pentru vectorul T. Avem:

Rearanjând, obținem

Știm că T este un vector unitate. Așadar, avem:

deci

Înlocuind în relația anterioară, avem:

Simplficând, avem:

Din care reiese că:

Observăm că poate avea două valori, din moment ce sunt 2 vectori în plan care au unghiul la . Totuși, știm că vectorul refractat va avea aceași direcție ca **I**, așa că vom cere o valoare pozitivă pentru , pentru a satisface

Deci, cu și pozitive, vrem să avem și pozitiv, deci vom lua

Subtituind în , obținem

+

Rezolvăm pentru :

Avem două soluții, din care alegem

Prin urmare, formula pentru vectorul refractat R este:

Vom folosi acum ecuația pentru R și o vom combina cu punctul de intersecție pentru a forma o nouă rază.

Apoi vom urmări raza prin cadru, și o vom înmulți cu coeficientul de refracție pentru a obține componenta refractată în punctul de intersecție.

Trebuie să ținem cont că în ecuația pentru R, I² poate să fie și complex. Acest lucru se întâmplă datorită fenomenului de reflexie internă.

Dacă raza de lumină este aproape tangentă la suprafață, ea poate fi reflectată total, chiar dacă suprafața are o componentă refractivă. Bijutierii folosesc acest lucru pentru a produce pietre prețioase care strălucesc în diferite feluri.

În cazul în care I² are o componentă imaginară, vom pretinde că reflexia este speculară și nu refractivă, găsim vectorul de reflecție și continuăm că mai sus.

# 4. Ray-Tracing distribuit în rețea

## 4.1 Introducere

Așa cum am discutat anterior, tehnica ray tracing este foarte costisitoare din punct de vedere computațional, fiind necesari timpi mari de execuție pentru a randa scene mai complexe, cu multe obiecte.

Pentru a îmbunătăți timpii de execuție, vom propaga munca în rețea folosind un model server - client, în care un calculator ales server va aștepta conexiuni disponibile, iar pentru fiecare conexiune realizată va trimite clientului respectiv părți din scenă pentru ca acesta să le proceseze.

Atunci când toate părțile din scenă vor fi procesate, server-ul va avea sarcina de a crea imagina din bucățile trimise de clienți și de a salva imaginea.

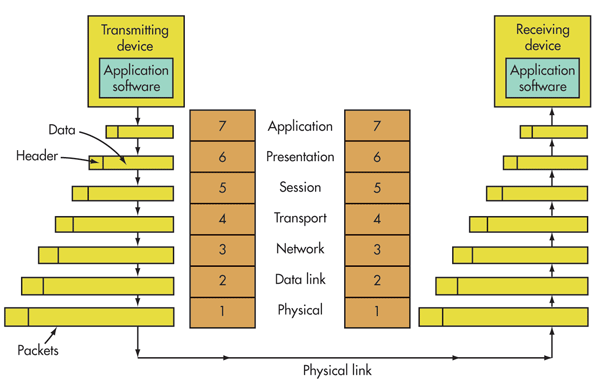
## 4.3 Rețeaua

O rețea constă din 2 sau mai multe calculatoare care sunt legate între ele pentru a partaja resurse, realiza transfer de fișiere sau pentru comunicații electronice. Calculatoarele dintr-o rețea pot fi legate între ele prin cabluri, linii de telefonie, unde radio sau sateliți. Două tipuri de rețea foarte des întâlnite sunt : LAN (rețea locală) și WAN (rețea extinsă).

Un protocol este un set de reguli care guvernează comuncarea dintre calculatoarele prezente într-o rețea. Modelul de referință OSI definește 7 straturi de protocoale de rețea. Modelul de Referință OSI (OSI este un acronim pentru interconectarea sistemelor deschise), pe scurt: OSI, este o stiva de protocoale de comunicație ierarhic foarte des folosit pentru a realiza o rețea de calculatoare. OSI este un standard al Organizației internaționale de standardizare, emis în 1984.

## 4.2.1 Modelul de referință OSI

Modelul de Referință OSI oferă metode generale pentru realizarea comunicației, sistemelor de calcul pentru ca acestea să poată schimba informații, indiferent de particularitățile constructive ale sistemelor (fabricant, sistem de operare, țară, etc). Modelul de Referință are aplicații în toate domeniile comunicațiilor de date, nu doar în cazul rețelelor de calculatoare.



Modelul OSI divizează problema complexă a comunicării între două sau mai multe sisteme în 7 straturi numite și niveluri (layers) distincte, într-o arhitectură ierarhică. Fiecare strat (nivel) are funcții bine determinate și comunică doar cu straturile adiacente. Cele 7 niveluri ale Modelului de Referință se numesc: Aplicație (nivelul 7, superior) , Prezentare, Sesiune, Transport, Rețea, Legătură de date, Fizic (nivelul 1, inferior).

Nivelul fizic definește specificații electrice, mecanice, procedurale și functionale pentru activarea, menținerea și dezactivarea legăturilor fizice între sisteme.Are rolul de a transmite unui șir de biți pe un canal de comunicații. Se precizează modulații, codări, sincronizări la nivel de bit.

Nivelul legatură de date se ocupă cu adresarea fizica, topologia rețelei, accesul la rețea, detecția și anunțarea erorilor și controlul fluxului fizic (flow control).Are rolul de a furinza un transport sigur, fiabil, al datelor de-a lungul unei legături fizice, realizând c ontrolul erorilor de comunicație, controlul fluxului de date, controlul legăturii, sincronizarea la nivel de cadru.

Nivelul rețea are roulul de determinare a căii optime pentru realizarea transferului de informații într-o rețea constituită din mai multe segmente, prin fragmentarea și reasamblarea informației.

Nivelul transport asigură transferul fiabil al informației între două sisteme terminale (end points) ale unei comunicații. Furnizează controlul erorilor și controlul fluxului de date între două puncte terminale, asigurând ordinea corectă a pachetelor de date. Oferă un serviciu de transport de date care izolează nivelurile superioare de orice specificitații legate de modul în care este executat transportul datelor.

Nivelul sesiune furnizează controlul comunicației între aplicații. Stabilește, menține, gestionează și închide conexiuni (sesiuni) între aplicații.

Nivelul prezentare transformă datele în formate înțelese de fiecare aplicație și de calculatoarele respective, compresia datelor și criptarea. Nivelul aplicație realizează interfața cu utilizatorul și interfața cu aplicațiile, specifică interfața de lucru cu utilizatorul și gestionează comunicația între aplicații. Acest strat nu reprezintă o aplicație de sine stătătoare, ci doar interfața între aplicații și componentele sistemelui de calcul.

Nivelul aplicație realizează interfața cu utilizatorul și interfața cu aplicațiile, specifică interfața de lucru cu utilizatorul și gestionează comunicația între aplicații. Acest strat nu reprezintă o aplicație de sine stătătoare, ci doar interfața între aplicații și componentele sistemelui de calcul. Un exemplu de asemenea interfață este dată de protocolul HTTP.

Deși astăzi există și alte sisteme, cei mai mulți distribuitori de echipamente de comunicație folosesc OSI pentru a instrui utilizatorii în folosirea echipamentelor. Se consideră că OSI este cel mai bun mijloc prin care se poate face înțeles modul în care informația este trimisă și primită.

**4.2.2 Sockeți**

Când ne referim la sockeți în contextul unei rețele, ne referim la ei ca fiind un punct terminal din cadrul unei cominicări între 2 noduri. O conexiune (sau o pereche de sockeți) constă într-o pereche de adrese IP care comunică între ele și intr-o pereche de porturi, unde un port este un număr pozitiv pe 32 de biți, de obicei reprezentat în formă decimală. Unele porturi sunt cunoscute și indică tipul serviciului către care se dorește conexunea.

Pentru multe aplicații mediul TCP/IP se așteaptă ca aplicațiile să folosească porturi cunoscute pentru a comunica între ele. Astfel, aplicațiile client pornesc de la premisa că server-ul va asculta portul respectiv asociat aplicației. De exemplu, portul pentru HTTP, protocolul folosit pentru a transmite pagini HTTP în www (Word Wide Web) este portul TCP 80. Ca și standard, un browser web ca încerca să deschidă o conexiune către portul 80 al hostului destinație, dacă nu cumva alt port este specificat în url.

Un port indentifică un punct de conexiune în stiva locală. Un socket identifică o adresă IP și un port (de exemplu 192.168.1.20:80 ar fi portul pentru server-ul web al hostului 192.168.1.20). Un socket poate fi considerat o asemenea pereche. O pereche socket conține 4 componente: adresa sursei, portul sursei, adresa destinației, portul destinației. Protocolul TCP permite conexiuni multiple pe același port atâta timp cât adresele IP care stabilesc conexiunea sunt diferite.

Porturile sunt împărțite în 3 intervale:

- porturile de la 0 la 1023 sunt cunoscute și asociate unui anumit serviciu

- porturile de la 1024 la 49151 sunt înregistrate și folosite pentru scopuri multiple

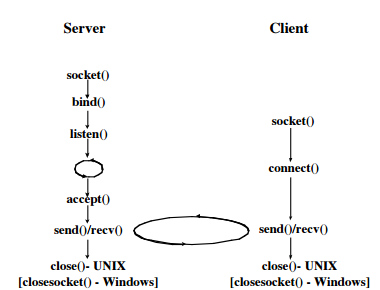
- porturile de la 49152 până la 65535 sunt porturi dinamice și private și nu ar trebui fie asociate niciunui serviciu

Prezentăm câteva servicii și porturile asociate lor:

|  |  |
| --- | --- |
| **Serviciu** | **Port** |
| DNS | 53 |
| HTTP/HTTPS | 80/443 |
| FTP | 20/21 |
| TELNET | 23 |
| POP3 | 110 |
| SMTP | 25 |
| NTP | 123 |

**4.2.2 Modelul server - client**

Modelul server – client este cel mai des întâlnit model folosit în rețele. Programul care inițiază transmisiunea este deseori referit ca client. Pe partea cealaltă, serverul este un program care așteaptă conexiuni de la clienți.



Aplicațiile server se încarcă în general la inițializarea sistemului de operare și ascultă conexiuni care ar putea să vină pe portul specificat. Aplicațiile client vor încerca să se conecteze la server având loc un schimb de informații. Când sesiunea este completă, de obicei clientul este cel care închide conexiunea.

**4.3 Programare asincronă**

În continuare vom prezenta conceptul de programare asincronă, care ne va folosi pentru a înțelege mai bine implementarea unui model de server care folosește comunicare asincronă.

Definim un fir de execuție ca fiind cea mai mică unitate de procesare ce poate fi programată spre execuție de către sistemul de operare. Este folosit în programare pentru a eficientiza execuția programelor, executând porțiuni distincte de cod în paralel în interiorul aceluiași proces.

Vom prezenta două modele de programare pe care le vom compara ulterior cu modelul asincron. Primul este modul sincron pe un singur fir de execuție :



Este un stil foarte simplu și intuitiv de a realiza un program care trebuie să realizeze diferite sarcini. Fiecare sarcină este realizată pe rând, fiecare sarcină terminându-se complet înainte ca programul să treacă la următoarea. De asmenea, dacă sarcinile sunt tot timpul realizate într-o ordine bine definită, implementarea unei sarcini mai din față pleacă de la premisa că toate sarcinile precedente au fost realizate cu succes, fără erori, datele acestora de ieșire putând fi folosite.

Vom prezenta acum modelul sincronizat care folosește mai multe fire de execuție.



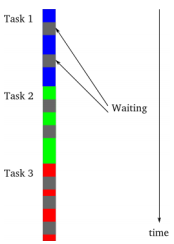
În acest model, programatorul își împarte logica programului în blocuri de cod care pot rula simultan. Însă, acest lucru este influențat considerabil de către sistemul de operare prezent și de hardware-ul sistemului, fiind posibil ca toate sarcinile să meargă în paralel, firele de execuție fiind total concurente, sau ca toate firele de execuție să fie intercalate împreună datorită limitărilor sistemului pe care este rulată aplicația. De menționat pentru acest model sunt problemele de sincronizare ale firelor de execuție cât și problema comunicării între ele (acolo unde este nevoie). Pentru sincronizarea firelor de execuție, avem la dispoziție mai multe metode, în funcție și de sistemul de operare. Pentru sincronizarea firelor de execuție avem la dispoziție mutexi, variabile condiționale și semafoare atât pe sisteme Posix cât și pe Windows. În plus, pe Windows mai avem la dispoziție secțiuni critice (excludere mutuală în cadrul aceluiași proces), evenimente și timere.

În modelul de programare asincron standard, sarcinile sunt intercalate între ele pe singur fir de execuție. Într-o implementare sincronă cu mai multe fire de execuție pe un sistem care nu poate folosi mai mult de un singur fir, sarcinile vor fi efectuate într-un mod intercalat însă programatorul își va reprezenta procesul de realizare al sarcinilor ca în figura anterioară, ceea ce ar putea cauza probleme de sincronizare sau de concurență în momentul în care aplicația ar fi rulată pe un sistem care dispune de mai multe procesoare. În schimb, o implementare asincronă care ar rula pe un singur fir de execuție va executa tot timpul sarcinile intercalate între ele, chiar și pe un sistem cu mai multe procesoare.

Într-o implementare sincronă pe mai multe fire de execuție, decizia de a suspenda un fir și de a executa un altul este în mare parte în afara controlului programatorului. Controlul aparține sistemului de operare, iar programatorul trebuie să presupună că un fir de execuție poate fi suspendat și înlocuit cu un altul în aproape orice moment. În schimb, în implementarea asincronă o sarcină va continua să ruleze până când se termină sau până când se renunță temporar în mod explicit la ea în vederea continuării altor sarcini.

O problemă în implementările asincron survine în cazul sarcinilor dependente între ele, acestea vor trebui să fie concepute astfel încât să poată primi date de intrare parțiale furnizate de celălalte sarcini de care depinde.

Din moment ce nu există paralelism, la prima vedere pare că un program ce utilizează implementare asincronă va dura la fel de mult ca un program ce utilizează o implementare sincronă pe un singur fir. Însă, în situația în care unele sarcini sunt forțate să aștepte, fiind blocante, modelul asincron este vizibil mai performant.



În acestă figură, secțiunile colorate în gri reprezintă perioade de timp când o sarcină așteaptă (este blocantă) și nu poate face niciun progres. Astfel de situații sunt întâlnite la efectuarea de operații de citire / scriere, transfer de date pe alte dispozitive sau în rețea, sau pur si simplu sarcini ce necesită un calcul computațional foarte complex. În general, un procesor este capabil să realizeze rate de transfer mult mai mari decât pot permite o rețea sau un dispozitiv de stocare. Din acest motiv, un program sincron care realizează multe operații de intrare / ieșire va pierde mult timp atunci când va avea de transferat date într-o rețea sau cu un alt dispozitiv de stocare. Din acest motiv, numim un astfel de program a fi blocant.

Ideea fundamentală a unui model asincron este de a executa alte sarcini atunci când întâlnește o sarcină blocantă. Astfel, un sistem asincron va bloca doar când nicio sarcină nu este capabilă să mai progreseze (din acest motiv un sistem asincron este adesea numit non-blocant). Fiecare trecere de la o sarcină la alta se întâmplă atunci când o sarcină se termină sau când ajunge într-un punct în care ar bloca. La un număr ridicat de sarcini blocante, un sistem asincron va fi mai performant decât unul sincron deoarece va petrece mai puțin timp așteptând, dedicând o cantintate de timp aproximativ egală fiecărei sarcini individuale efectuate.

Modelul asincron se comportă mai bine decât cel sincron în anumite situații:

* + există un număr mare de sarcini deci este foarte probabil să existe întotdeauna o sarcină care sa poată face progres
  + sarcinile trebuie să realizeze multe operații de intrare / ieșire, făcând un program implementat în mod sincron să piard mult timp blocând alte sarcini care ar putea să fie efectuate în acel timp
  + sarcinile sunt independente unele față de celălalte, deci nu este necesar ca o sarcină să aștepte după altele

Aceste condiții caracterizează un server de rețea într-un mediu server – client. Fiecare sarcină reprezintă o conexiune de client care trebuie să efectueze operații de intrare – ieșire în rețea.

O posibilă alternativă este utilizarea câte unui fir de execuție pentru fiecare conexiune de client. Însă pe măsură ce numărl de fire de execuție crește, serverul poate să aibă probleme de performanță deoarece fiecare fir presupune consum de memorie suplimentar asociat cu crearea și menținerea stării acestuia.

Evitarea schimbărilor de context este un alt avantaj al modelui asincron de programare. De fiecare dată când sistemul de operare transferă controlul de la un fir de execuție către altul este nevoit să actualizeze starea firului de execuție curent pentru a se putea întoarce la el mai târziu.

**Bibliografie:**

1. Rober L.Cookand Kenneth E.Torrance. A reﬂectance model forcomputer graphics. SIGGRAPH Comput. Graph. 15:307–316, August 1981.
2. Andrew S. Glassner An introduction to ray tracing. Academic Press Ltd., London, UK, 1989.
3. Richard Hacker. Certiﬁcation of algorithm 112: Position of point relative to polygon. Commun. ACM, 5:606–, December 1962.
4. Gernot Hoffmann. Cie color space. http://www.fhoemden.de/˜hoffmann/ciexyz29082000.pdf.
5. F.A. Jenkins and H.E. White. Fundamentals of Optics. McGraw-Hill, 2001.
6. Bui Tuong Phong. Illumination for computer generated pictures. Commun. ACM, 18:311–317, June 1975.
7. M.Shimrat. Algorithm112: Position of point relative to polygon. Commun.ACM, 5:434–, August 1962.
8. Michael Stokes, Matthew Anderson, SrinivasanChandrasekar and RicardoMotta. A standard default color space for the internet
9. Technical Report 1.10, International Color Consortium, Hewlett-Packard, and Microsoft, 1995.
10. <http://www.comp.hkbu.edu.hk/~comp2330/lecture/notes/C-Sockets.pdf>
11. <http://boost.cowic.de/rc/pdf/asio_doc.pdf>
12. [ITU-T X.200 (the same contents as from ISO)](http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.200-199407-I!!PDF-E&type=items)
13. [The ISO OSI Reference Model , Beluga graph of data units and groups of layers](http://infchg.appspot.com/usr?at=1263939371)
14. Zimmermann, Hubert (April 1980). "OSI Reference Model — The ISO Model of Architecture for Open Systems Interconnection". *IEEE Transactions on Communications* **28**(4): 425–432. [doi](http://en.wikipedia.org/wiki/Digital_object_identifier):[10.1109/TCOM.1980.1094702](http://dx.doi.org/10.1109%2FTCOM.1980.1094702). [CiteSeerX](http://en.wikipedia.org/wiki/CiteSeer#CiteSeerX): [10.1.1.136.9497](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.136.9497).
15. Computer Networks, Andrew Tanenbaum, Second edition ©1989 Prentice Hall. [covers networking protocols and theory] Connected: An Internet Encyclopedia, Third Edition. IP Addressing Review, http://www.freesoft.org/CIE/Course/Subnet/1.htm