Cursul
Sisteme de prelucrare grafica

Curs: Prof. dr. ing. Florica Moldoveanu

Laborator si proiect:
Conf. dr. ing. Anca Morar
S.l. dr. ing. Alexandru Grădinaru
Asist. drd. inf. Andrei Lambru
Asist. drd. inf. Anca Bălețoiu
Asist. drd. inf. Andrei Lăpușteanu

2021 - 2022

Echipa SPG

- **Face parte din grupul de "Grafică şi Realitate Virtuală 3DUPB" din departamentul de Calculatoare (https://3d.pub.ro/)**
- **Activitati de cercetare** in domeniile: Grafica pe calculator (Computer Graphics), Sisteme cu viziune (Computer Vision), Realitate Virtuala si Realitate Augmenta cu diverse aplicatii, in special in domeniul medical.
  - Numeroase proiecte de cercetare (https://3d.pub.ro/projects.html)

    TRAVEE – recuperare neuromotorie bazata pe feedback augmentat în realitate virtuala.
  - Ultimul proiect important: **Sound of Vision** (https://soundofvision.net/)
    - Finalizat cu realizarea unui dispozitiv portabil dedicat nevazatorilor, pentru a-i ajuta sa se deplaseze liber in mediul inconjurator
- **Activitati didactice**: EGC, SPG, Ingineria Programelor, cursuri la programul de masterat GMRV şi MTI.
- **Coordonare doctoranzi**

Structura cursului

Partea I-a:

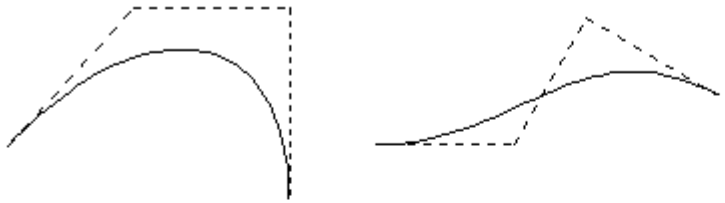
- **Notiuni avansate de grafică 3D**: modelarea şi vizualizarea curbelor şi a suprafetelor de forma libera, metode de texturare a suprafetelor, animația bazată pe schelet.
- **Tehnici de redare fotorealista folosind shadere (**OpenGL3.x si GLSL) - laborator
- Aplicații:
  - Sisteme de Proiectare Asistata de Calculator pentru arhitectura, design industrial si artistic
  - Spatii virtuale 3D pentru e-Learning, e-Commerce, divertisment, socializare, etc.
  - Jocuri şi alte aplicatii grafice 3D pentru calculatoare desktop şi dispozitive mobile

Partea a II-a:

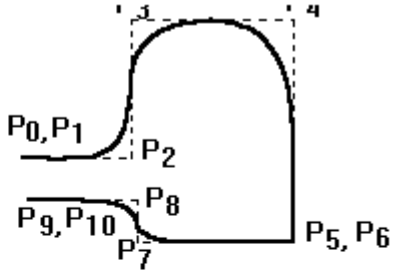
Notiuni şi algoritmi fundamentali de prelucrare şi analiză a imaginilor digitale.

- Aplicații:
  - Analiza imaginilor medicale 2D si 3D
  - Recunoasterea obiectelor din imagini
  - Analiza imaginilor satelitare
  - Recunoasterea fețelor
  - Recunoasterea scrisului
  - Robotică
  - Vehicule autonome, etc.

Cap. 1 şi 2: Modelarea şi vizualizarea curbelor şi a suprafetelor de forma libera

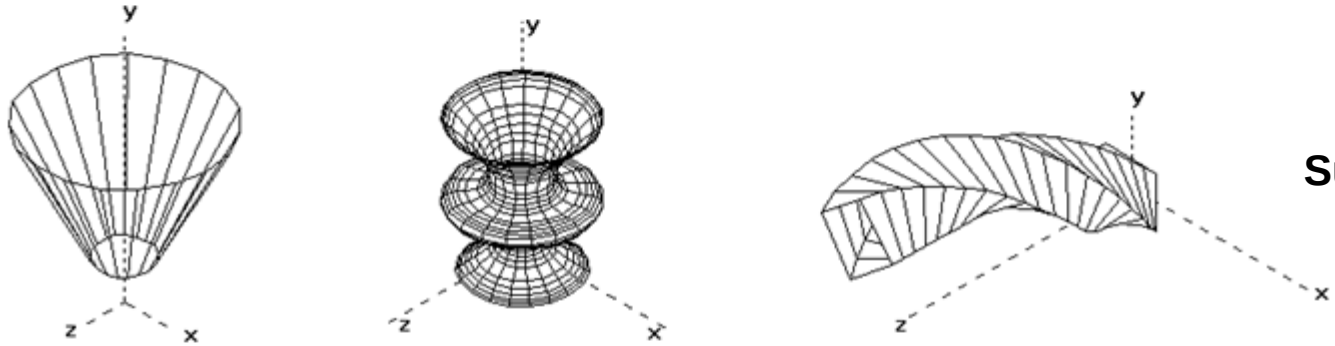


Curbe Bezier



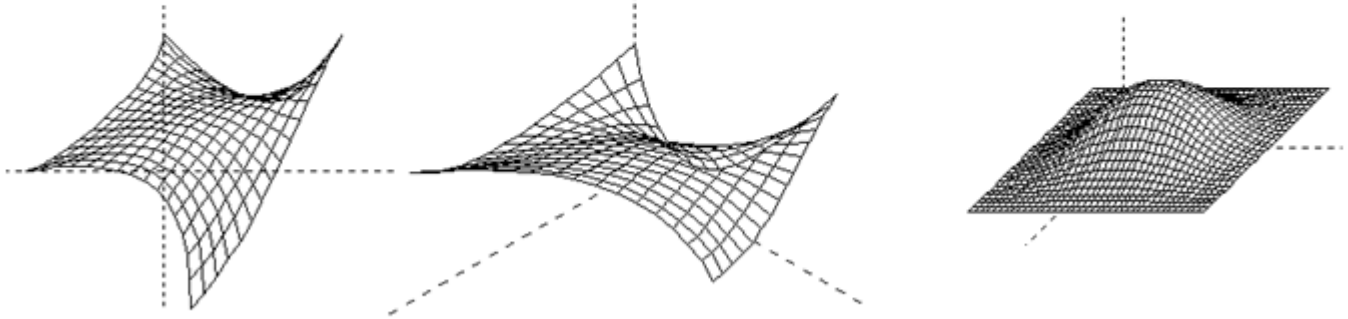
Curba B-spline

Curbe de forma libera



Suprafete de rotatie

Suprafete de rotatie si translatie.



Suprafete Bezier

Suprafata B-spline

Suprafete de forma libera

Modelarea şi vizualizarea curbelor şi a suprafetelor de forma libera
- utilitate -

Multe forme din lumea reala sunt “de forma libera” - nu pot fi modelate prin ecuatii cunoscute din geometria analitica

Noţiunile sunt utile pentru:

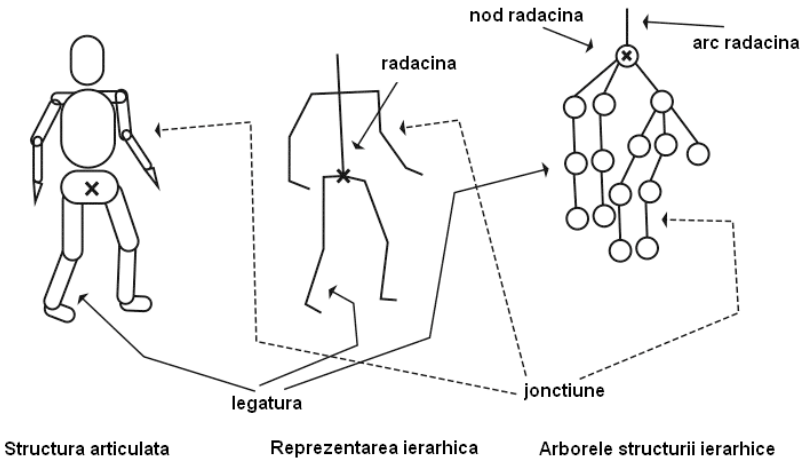
- Modelarea suprafeţelor cu geometrie complexă: obiecte de mobilier, de arta, caroseria automobilelor, forma avioanelor, etc.
- Realizarea animaţiei pe calculator (traiectoria spatiala a unui obiect – curba de forma libera)
- Sisteme GIS (reprezentarea mai eficienta a formei strazilor, a formelor de relief, a curbelor de nivel)
- Reprezentarea formelor extrase din imagini, ş.a.

Cap.3: Tehnici avansate de redare a scenelor 3D prin folosirea texturilor

- Metode de aplicare a texturilor pe suprafetele 3D
- Simularea detaliilor suprafetelor:
  - Normal mapping
  - Parallax mapping
- Redarea mediului înconjurator pe suprafetele obiectelor prin simularea reflexiei si a refractiei:
  - Environment mapping (Cube mapping)

Cap.4: Animația bazată pe schelet (Skeletal animation)

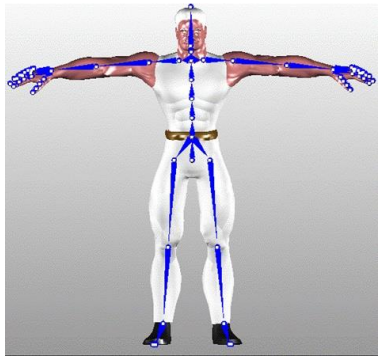
- Modelarea structurilor ierarhice articulate
- Animarea structurii articulate
- Cinematica directă – definirea de cadre cheie

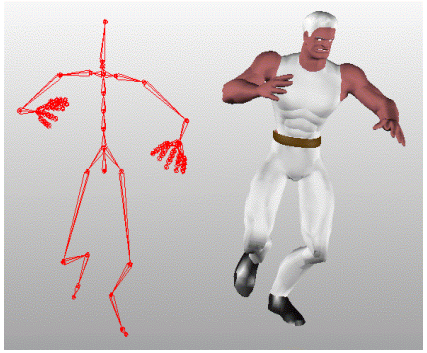


- "Skinning" (Skeleton Subspace Deformation)

  "schelet" (structura articulata) + "piele" (mesh)

  - Legarea vârfurilor mesh-ului la "oase"
  - Transformarea vârfurilor mesh-ului folosind transformarile asociate oaselor

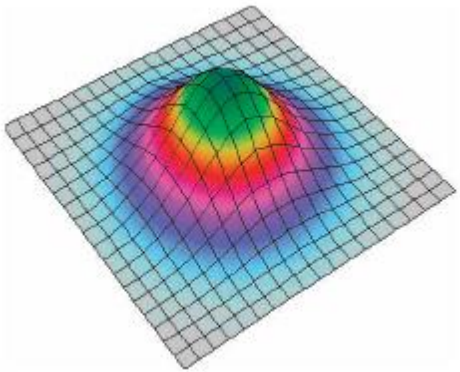




Cap. 5: Tehnici de restaurare şi imbunatatire a imaginilor digitale

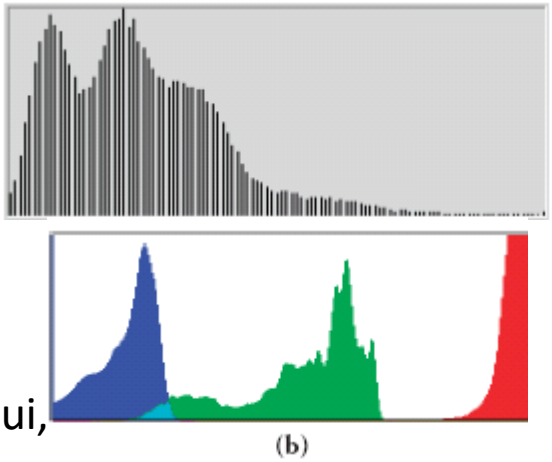
Scop:

- Reducerea defectelor introduse în imagini de dispozitivele de achiziție
- Îmbunatatirea perceptiei informatiei continute în imagini



Metode:

- Reducerea zgomotelor (informație falsă în valorile pixelilor) folosind diferite tipuri de filtre; corectia distorsiunilor geometrice.



- Transformarea nivelurilor de gri pentru îmbunatatirea contrastului, transformarea histogramei.

Cap.6: Extragerea caracteristicilor din imagini (linii si frontiere), bazata pe punctele de frontiera

- Detectori de pixeli de frontiera bazati pe Gradientul imaginii
- Detectori pixeli de frontiera bazati pe Laplacianul imaginii
- Detectorul de frontiere Canny
- Transformata Hough (pentru detectia liniilor drepte si a curbelor parametrice)

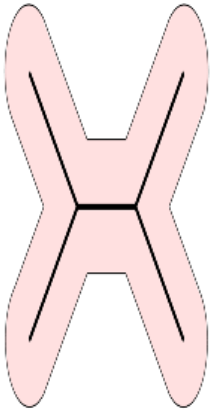
Cap.7: Segmentarea imaginilor bazată pe regiuni

7.1. Agoritmi de segmentare: grupeaza pixelii adiacenti cu atribute similare în regiuni, fiecare regiune avand un identificator unic.

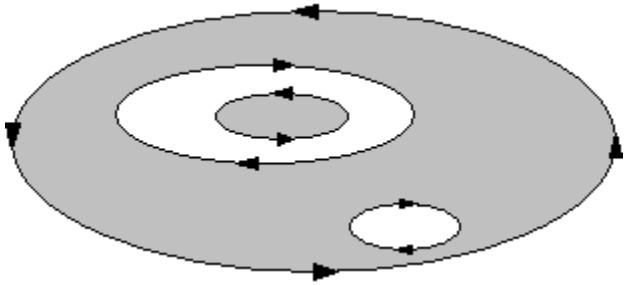
Atribute la nivel de pixel folosite in segmentarea bazata pe regiuni: culoarea, intensitatea, adancimea, ş.a

- Segmentarea bazata pe regiuni vs. segmentarea semantică (atribuie fiecare pixel unei clase)

7.2. Algoritmi de subtiere si scheletizare (recunoasterea unor forme pe baza "scheletului" lor)



7.3. Extragerea contururilor regiunilor



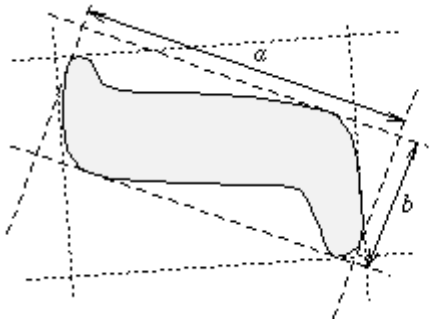
Cap.8: Descrierea si analiza formelor extrase din imagini

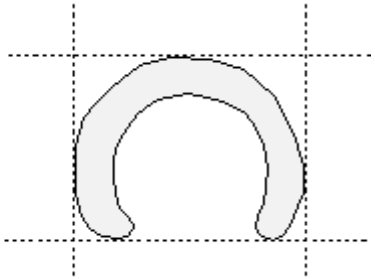
8.1. Analiza caracteristicilor locale ale frontierelor - reprezentarea prin coduri de inlantuire

8.2. Aproximarea poligonala a frontierelor

8.3. Recunoasterea formelor din imagini:

- **Analiza bazata pe caracteristicile conturului**
- **Analiza bazata pe caracteristicile topologice ale suprafatei**

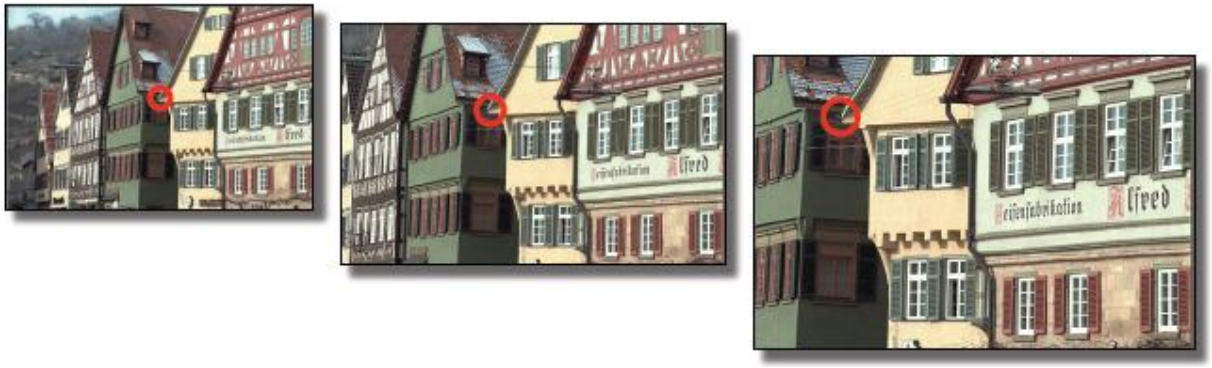




Cap.9: Recunoasterea imaginilor prin detectia caracteristicilor locale

- O imagine poate fi recunoscuta pe baza unui set de caracteristici locale.
- O **caracteristica locala într-o imagine** este un şablon de intensitati/culori care se diferenţiaza de vecinatatea sa. Trebuie sa fie invarianta la transformari afine, schimbari de luminozitate, directie de iluminare.

9.1. Extragerea caracteristicilor locale din imagini: puncte cheie, spatiul de scalare



9.2. Reprezentarea caracteristicilor locale prin descriptori: SIFT

Tematica laborator

- **Aplicatii folosind OpenGL 3.x si GLSL:**
  - Geometry shader – implementare si utilizare
  - Modelare si vizualizare curbe şi suprafete de forma libera
  - Realizarea de efecte fotorealiste prin operatii în banda grafica, folosind reflexii si texturi
  - Sisteme de particule (pe GPU)
  - Redarea în timp real prin “deferred rendering”
- **Implementarea unor algoritmi de îmbunatatire a imaginilor digitale**
- **Implementarea unor algoritmi de extragere a frontierelor din imagini, prin detectia punctelor de frontiera**

CURBE si SUPRAFETE de Forma Libera

Prof. univ. dr. ing. Florica Moldoveanu

Curbe si suprafete de forma libera

- **Se folosesc pentru reprezentarea suprafetelor obiectelor** cu geometrie complexa: caroseria unui avion, a unei nave, a unui automobil, suprafata unei cesti de cafea, etc.
- **Modelul geometric** al unei curbe/suprafete de forma libera **este o expresie matematica** prin care se pot calcula puncte de pe curba/suprafață.
- **Algoritmii de redare discretizeaza curba/suprafata** calculand puncte de pe ea pe baza modelului geometric;
  - punctele se folosesc pentru a aproxima curba printr-o polilinie, suprafata prin fațete plate
  - în cazul unei suprafețe, se calculeaza si vectorul normal în fiecare punct, pentru simularea reflexiei/refractiei luminii in acel punct.

Cum sunt definite?

- **Date geometrice**:

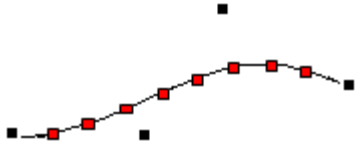
  - **puncte din plan** (curbe 2D) **sau din spaţiu** (curbe şi suprafete 3D)
  - în unele cazuri, tangente în punctele respective si alte conditii geometrice

- Setul de puncte folosit pentru redarea curbei/suprafetei se calculeaza folosind datele geometrice si modelul geometric al curbei/suprafetei.

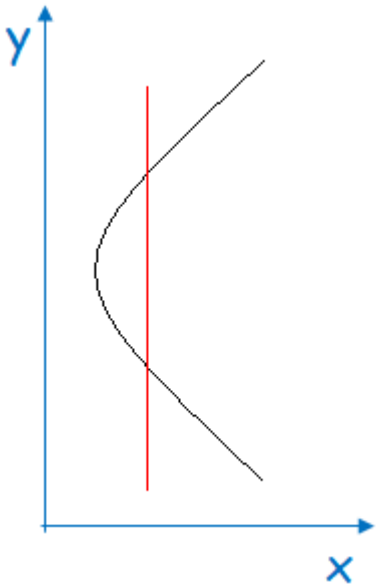
- 2 tipuri de curbe /suprafete:

- **De interpolare**: punctele date (care definesc geometria) se afla pe curbă / suprafață – punctele folosite la redare se obtin prin interpolare intre punctele date

- **De aproximare**: punctele date controleaza forma curbei/suprafetei - nu apartin in mod necesar curbei/suprafetei.



Cum sunt modelate



Ec. explicite?

- Curba 2D: $y = f(x)$; curba 3D: $y = f(x)$, $z=g(x)$
- Suprafata 3D: $z=f(x,y)$

→ Nu pot reprezenta puncte multiple: la o val a lui x→ o singura valoare lui y

→ Dependente de sistemul de coordonate

→ Rar folosite in Computer Graphics

Ec. implicite?

- Curba 2D: $f(x,y) = 0$ → $ax+by+c =0$; $x^2+y^2-r^2 = 0$
- Suprafata 3D: $f(x,y,z) = 0$ → $ax + by + cz + d = 0$; $x^2 + y^2 + z^2 - r^2 = 0$ → **Utile in unele cazuri;**

  ex: $f(x,y,z)$ poate reprezenta o suprafata inchisa: $f(px,py,pz) < 0$ →p este punct interior suprafetei; $f(px,py,pz)=0$→punct pe suprafata; $f(px,py,pz) > 0$-> punct exterior

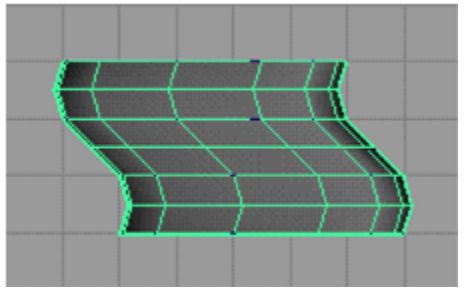
Dezavantaj: nu reflecta relatia dintre forma curbei/suprafetei si datele prin care este definita

- Ec. explicite si implicite din geom. analitica modeleaza un numar mic de curbe si suprafete.
- Formele geometrice complexe nu sunt cunoscute prin functii analitice.
- **Curbele si suprafetele de forma libera sunt modelate prin ecuatii parametrice.**

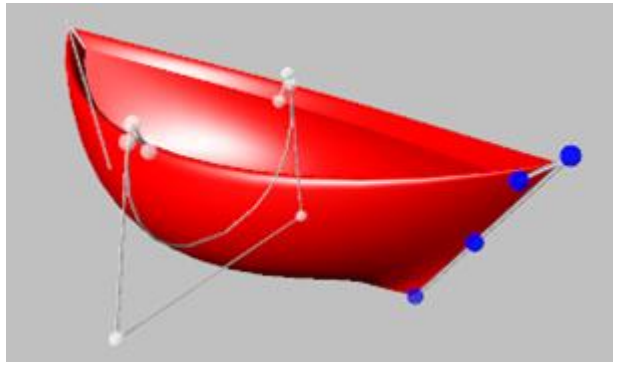
Istoric (1)

- Constructorii de nave au fost printre primii care au dezvoltat modele matematice pentru reprezentarea suprafetelor
  - Unele parti ale suprafetei unei nave sunt plate intr-o directie si curbe in cealalta directie; astfel de suprafete au fost denumite „lofted surfaces" (Loftsman: o persoana care reproduce in marime reala desenul unei nave sau al unui avion, lucrand pe podeaua unei cladiri)

Lofted surface



Tehnica "lofting" a fost folosita pentru prima data in proiectarea navelor.



Istoric (2)

- 1960-1970: constructorii de automobile si avioane au devenit interesati in folosirea calculatoarelor pentru automatizarea muncii de proiectare:
  - Pierre Bézier la Renault, Paul de Casteljau la Citroën, Steven Coons la Ford Motors
  - James Ferguson – modelarea de suprafete de avioane
- 1974: Computer assisted geometric design (CAGD)
- Anii 1980: au fost dezvoltate sisteme software CAGD de uz general in constructii dar si in alte domenii, ca: arhitectura (pt modelarea 3D a cladirilor), chimie (modelarea moleculelor), geostiinta (harti specializate), s. a.
- Anii 1990: sistemele CAGD au fost utilizate pt crearea de efecte speciale in filme sau chiar pentru generarea in intregime a filmelor (*Jurassic Park, Toy story, Finding Nemo, Shrek, si altele).*

Le Centre Pompidou – Metz, France,Shigeru Ban and Jean de Gastines, 2010

[photo by Nicolas Billiaux, 2009, www.flickr.com]

CURBE DE FORMA LIBERA - 1

Prof. univ. dr. ing. Florica Moldoveanu

Utilizari

Pentru forme neconvenţionale (care nu sunt cunoscute prin functii din geometria analitica: cerc, elipsa, s.a.):

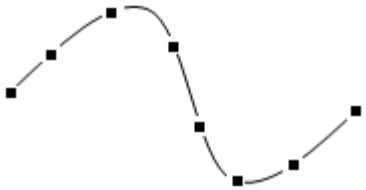
- desen artistic,
- arhitectura,
- caractere de text (fonturi),
- animație: traiectorii oarecare în spaţiu,
- definirea suprafețelor de formă liberă,
- descrierea formelor extrase din imagini,
- etc.

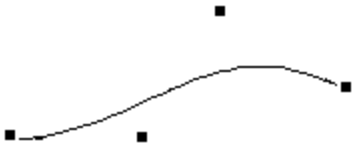
Cum sunt definite?

- Prin puncte din plan (curbe 2D) sau din spaţiu (curbe 3D)
- In unele cazuri se mai dau tangente în punctele respective

2 tipuri de curbe de forma libera:

- De interpolare: trec prin toate punctele date
- De aproximare: punctele date controleaza forma curbei





Ecuatiile parametrice ale unei curbe de forma libera

Reprezentarea parametrica:

x(u) = fx(u);
y(u) = fy(u); umin<= u <= umax
z(u) = fz(u);

fx, fy, fz : functii polinomiale

Exemple:

Un segment de dreapta:
x = x1 + u(x2 - x1)
y = y1 + u(y2-y1) --> x(u) = a1x* u + aox
y(u) = a1y*u + a0y

O curba cubica:

$X(u) = a_{3x}{}^{*}u^3 + a_{2x}{}^{*}u^2 + a_{1x}{}^{*}u + a_{0x}$

$Y(u) = a_{3y}{}^{*}u^3 + a_{2y}{}^{*}u^2 + a_{1y}{}^{*}u + a_{0y}$

$Z(u) = a_{3z}{}^{*}u^3 + a_{2z}{}^{*}u^2 + a_{1z}{}^{*}u + a_{0z}$

a_{ix}, a_{iy}, a_{iz} – **coeficientii algebrici ai curbei**

Reprezentarea parametrica vectoriala

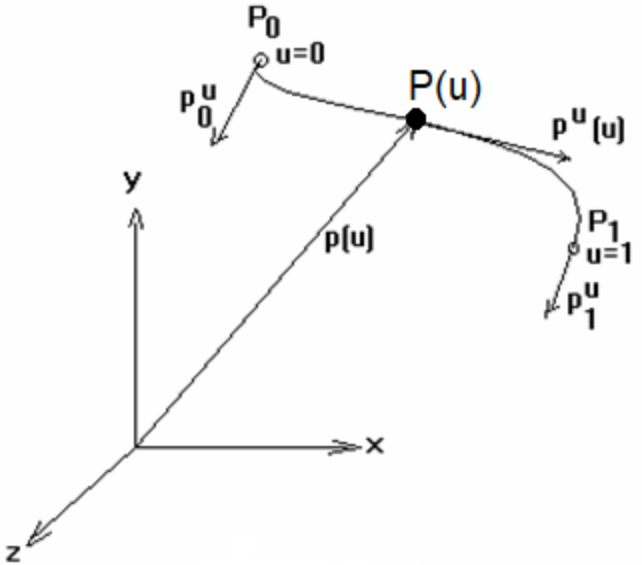
$p(u) = a_3u^3 + a_2u^2 + a_1u + a_0$

$0 <= u <= 1$

$p(u)$, a_0, a_1, a_2, a_3 – vectori de 2/ 3 componente

Ex : $p(u) = [x(u)\ y(u)\ z(u)]$
$a_0 = [a_{0x}\ a_{0y}\ a_{0z}]$

P(u) = (x(u), y(u), z(u)) - punct de pe curba



Prin ce se deosebesc diferitele tipuri de curbe de forma libera?

Reprezentarea prin coeficientii algebrici: "forma algebrică"

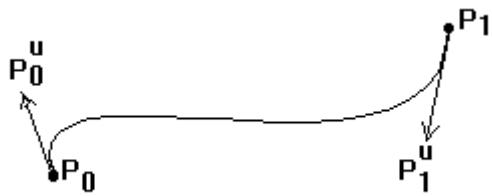
$$p(u) = a_3u^3 + a_2u^2 + a_1u + a_0$$

- NU evidentiaza diferenţele dintre diferite tipuri de curbe de forma libera
- Este aceeasi pentru orice curba (segment de curba) cubica
- Curbele de forma libera se deosebesc prin **condiţiile geometrice pe care le satisfac: datele geometrice prin care sunt definite.**
- Conditiile geometrice apar explicit în "**forma geometrică**"
- Forma geometrică reprezintă **modelul geometric**, specific fiecărui tip de curbă.

Curbe Hermite (1)

Datele geometrice prin care este definita o curba Hermite:

punctele extreme și tangentele în punctele extreme



Curba Hermite

- datele geometrice prin care este definita curba:

$P_0\,(P_0.x, P_0.y, P_0.z)$, $P_1\,(P_1.x, P_1.y, P_1.z)$, $P_0^u\,(dx0, dy0, dz0)$, $P_1^u\,(dx1, dy1, dz1)$

Obtinem forma geometrică plecand de la forma algebrică:

$$\mathbf{p(u) = a_3u^3 + a_2u^2 + a_1u + a_0}$$

$$\mathbf{0 <= u <= 1}$$

Se impun conditiile geometrice in forma algebrica →

$$p(0) = P_0$$

$$p(1) = a_3 + a_2 + a_1 + a_0 = P_1$$

$$p(u)^u = 3^*a_3{}^*u^2 + 2^*a_2{}^*u + a_1$$

$$p^u(0) = a_1 = P_0^u$$

$$p^u(1) = a_1 + 2^*a_2 + 3^*\ a_3 = P_1^u$$

Curbe Hermite(2)

- Se obtin expresiile coeficientilor algebrici, a_0, a_1, a_2, a_3, în functie de conditiile geometrice
- Se înlocuiesc coeficientii algebrici in forma algebrica, grupand termenii pt a evidentia conditiile geometrice.
- Rezulta **forma geometrică vectorială a curbelor Hermite**:

$$p(u) = (2u^3 - 3u^2 + 1)P_0 + (-2u^3 + 3u^2)P_1 + (u^3 - 2u^2 + u)P_0^u + (u^3 - u^2)P_1^u$$

Se introduc notatiile:

$F1(u) = 2^*u^3 - 3^*u^2 + 1$

$F2(u) = -2^*u^3 + 3^*u^2$

$F3(u) = u^3 - 2^*u^2 + u$

$F4(u) = u^3 - u^2$

Forma geometrica vectoriala a curbelor Hermite

$$\boxed{p(u) = F1(u)^*P_0 + F2(u)^*P_1 + F3(u)^*P_0^u + F4(u)^*P_1^u}$$

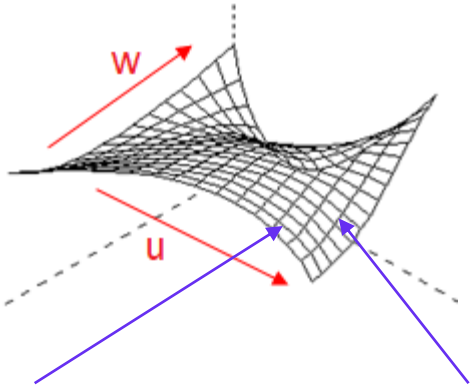
P_0, P_1, P_0^u, P_1^u sunt conditiile (datele) geometrice

F1, F2, F3, F4 – se numesc "functii de amestec" (blending functions) sau ”de ponderare” pentru conditiile geometrice: determina contributia conditiilor geometrice în calculul punctelor intermediare de pe curba

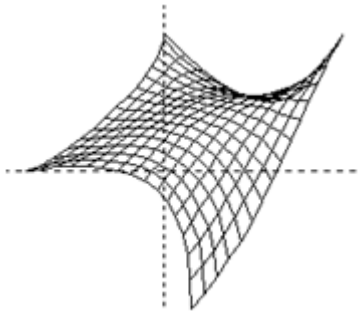
Curbe Hermite(3)

Formele matriciale

Formele matriciale permit extinderea directa a modelelor curbelor la suprafețe!



Orice petic de suprafaţă de forma libera, p(u,w) este alcatuit din 2 familii de curbe de forma libera ortogonale în spațiul parametric (u,w)!



Curbe p(u, wconst) Curbe p(uconst, w)

Forma algebrica vectoriala:

$$p(u) = a_3 * u^3 + a_2 * u^2 + a_1 * u + a_0$$

$$p = [u^3 \;\; u^2 \;\; u \;\; 1]\,[a_3 \;\; a_2 \;\; a_1 \;\; a_0]^T$$

$$U = [u^3 \;\; u^2 \;\; u \;\; 1]$$

$$A = [a_3 \;\; a_2 \;\; a_1 \;\; a_0]^T$$

$$\boxed{P = U * A \quad \text{Forma algebrica matriciala}}$$

Curbe Hermite(3)

Formele matriciale

Forma geometrica vectoriala:

$$p(u) = F1(u)^* P_0 + F2(u)^* P_1 + F3(u)^* P_0^u + F4(u)^* P_1^u$$

$$p = [F1 \;\; F2 \;\; F3 \;\; F4]\,[P_0 \;\; P_1 \;\; P_0^u \;\; P_1^u]^T$$

$$F = [F1 \;\; F2 \;\; F3 \;\; F4]$$

$$G_H = [P_0 \;\; P_1 \;\; P_0^u \;\; P_1^u]^T$$

$$p = F^* G_H$$

$G_H = [P_0 \;\; P_1 \;\; P_0^u \;\; P_1^u]^T$ **- vectorul conditiilor geometrice pentru o curbă Hermite**

$$F(u) = [F1(u) \;\; F2(u) \;\; F3(u) \;\; F4(u)] = [u^3 \;\; u^2 \;\; u \;\; 1]^* M_H$$

$$M_H = \begin{vmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{vmatrix}$$ **Matricea de bază Hermite**

$$\boxed{p = U^* M_H^* G_H \quad \text{Forma geometrică matricială a curbelor Hermite}}$$

Curbe Hermite(4)

Conversia de la forma geometrica la cea algebrica:

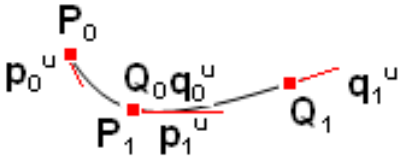
$p = U^* A \longleftarrow p = U^* M_H{}^* G_H \longrightarrow A = M_H{}^* G_H$

Compunerea curbelor Hermite

De ce este necesara? **- Pentru a obtine curbe care trec prin mai mult de 2 puncte si au proprietatile curbelor Hermite.**

Doua curbe Hermite, G1 şi G2: $G1 = [P_0 \;\; P_1 \;\; p_0^u \;\; p_1^u]^T \quad G2 = [Q_0 \;\; Q_1 \;\; q_0^u \;\; q_1^u]^T$

pot fi compuse daca sunt asigurate conditiile de continuitate geometrica de ordin 0 si 1 in punctul de jonctiune:



- P_1 coincide cu Q_0

- cele doua curbe au aceeasi tangenta in punctul de jonctiune

Curbe Hermite(4)

Afişarea curbelor definite parametric

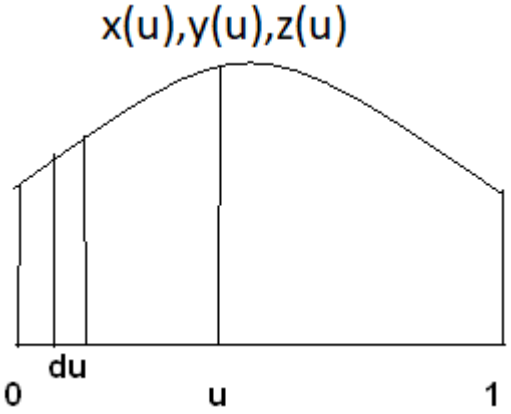
Se foloseste forma geometrica vectoriala:

$$p(u) = F1(u)^{*}P_0 + F2(u)^{*}P_1 + F3(u)^{*}\ P_0^{u} + F4(u)^{*}\ P_1^{u}$$

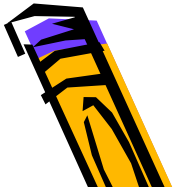
$p(u) = (x(u), y(u), z(u))$ $\qquad$ $0 <= u <= 1$

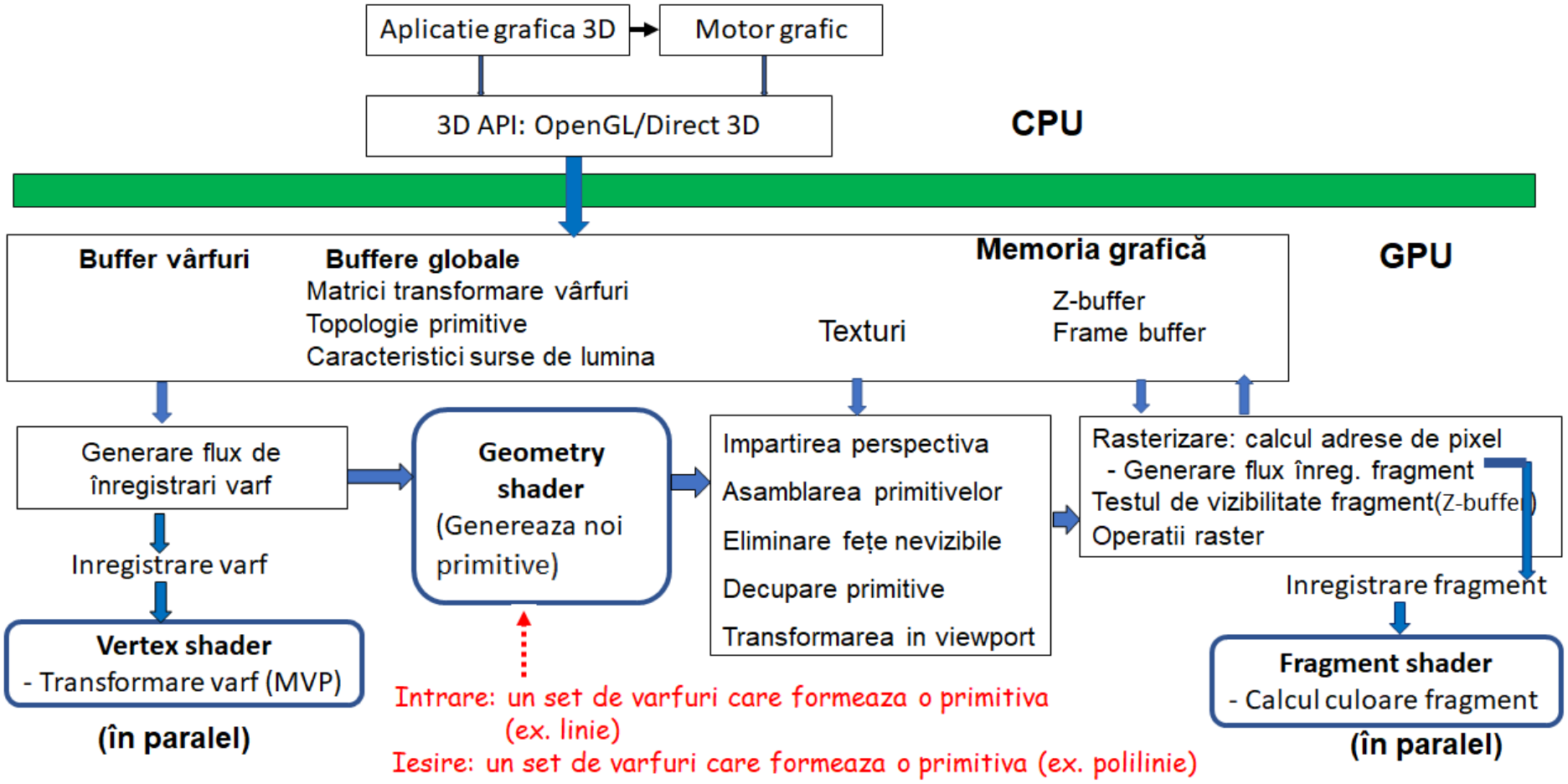
Se discretizeaza curba calculând puncte succesive de pe ea, cu un pas constant, si se afişează ca polilinie.

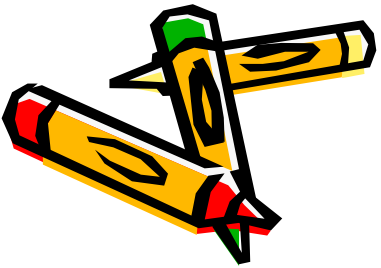
Eficient → calculul punctelor de pe curba sa se faca in Geometry shader



Geometry shader







Creaza geometrie: introduce in banda grafica primitive care nu au fost transmise din programul executat pe CPU!

CURBE DE FORMA LIBERA - 2

Prof. univ. dr. ing. Florica Moldoveanu

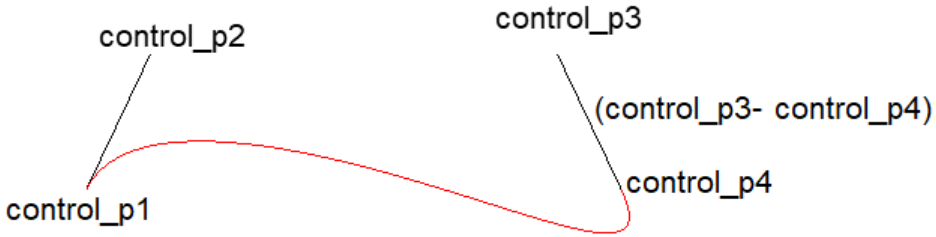
Afisarea curbelor Hermite folosind Geometry shader (1)

Pentru calculul punctelor de pe curba se foloseste forma geometrica vectoriala:

$$p(u) = F1(u)^{*}P_0 + F2(u)^{*}P_1 + F3(u)^{*}\ P_0^{u} + F4(u)^{*}\ P_1^{u}$$

Cele 4 conditii geometrice sunt transmise la GS prin variabile uniforme:

```
uniform vec3 control_p1, control_p2, control_p3, control_p4;
```



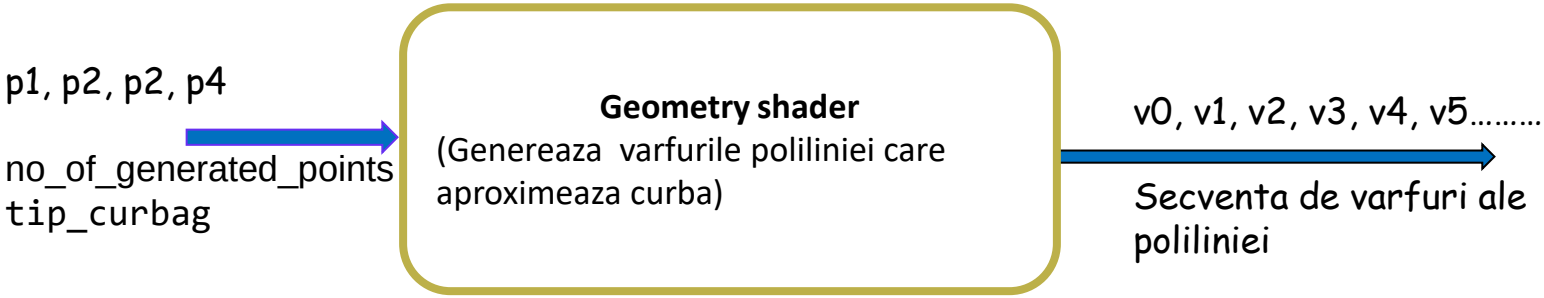
Curba afisata

Calculul punctelor de pe curba este implementat in functia Hermite care face parte din Geometry shader:

```
vec3 Hermite (float u)
{ //întoarce punctul de pe curba
  return (2*u*u*u - 3*u*u +1)*control_p1 + (-2*u*u*u + 3*u*u)*control_p4
         + (u*u*u - 2*u*u + u)*Tp1 + (u*u*u - u*u)*Tp4; }
```

Afisarea curbelor Hermite folosind Geometry shader (2)

```
uniform vec3 control_p1, control_p2, control_p3, control_p4;
uniform int no_of_generated_points;//numarul de puncte de pe curba, de generat
uniform int tip_curbag;//tipul curbei de generat: acelasi GS se foloseste pt
//afisarea mai multor tipuri de curbe
```



control_p1: punctul initial; control_p4: punctul final
control_p2, control_p3 : folosite in calculul tangentelor in p1 si p4

vec3 Tp1 = control_p2 - control_p1; // tangenta in punctul initial
vec3 Tp4 = control_p3- control_p4; // tangenta in punctul final

Afisarea curbelor Hermite folosind Geometry shader (3)

```
// Functia Init
{  ……————————————————————————————..
// Se definesc programele shader pentru generarea curbelor
{ Shader *shader = new Shader("CurveGeneration");
  shader->AddShader("Source/……/VertexShader.glsl", GL_VERTEX_SHADER);
  shader->AddShader("Source/……/GeometryShader.glsl", GL_GEOMETRY_SHADER);
  shader->AddShader("Source/…../FragmentShader.glsl", GL_FRAGMENT_SHADER);
  shader->CreateAndLink();
  shaders[shader->GetName()] = shader;
}
control_p1 = glm::vec3(-3, 0, 1.0); control_p2 = glm::vec3(-2, 2, 1.0);
control_p3 = glm::vec3(2,2, 1.0); control_p4 = glm::vec3(3,0, 1.0);
no_of_generated_points = 50;

// Creaza mesh cu o linie, care va fi intrare pt Geometry shader
{  vector<VertexFormat> vertices
    { VertexFormat(glm::vec3(-4.0, -2.5,  1.0), glm::vec3(0, 1, 0)),
      VertexFormat(glm::vec3(-4.0, 5.5,  1.0), glm::vec3(0, 1, 0))
    };
      vector<unsigned short> indices ={0, 1};
      meshes["curve"] = new Mesh("generated curve");
      meshes["curve"]->InitFromData(vertices, indices);
      meshes["curve"]->SetDrawMode(GL_LINES);
}
```

Afisarea curbelor Hermite folosind Geometry shader (4)

```
//Geometry shader (GS)
#version 330

layout(lines) in;//se declara tipul primitivei de intrare – aici nu se foloseste
layout(line_strip, max_vertices = 256) out;//iesirea din GS este o polilinie


uniform mat4 View, Projection;
uniform int tip_curbag;//tipul curbei de generat
uniform vec3 control_p1, control_p2, control_p3, control_p4;
uniform int no_of_generated_points;// numarul de puncte de pe curba, de generat

vec3 Tp1 = control_p2 - control_p1; // tangenta in punctul initial
vec3 Tp4 = control_p3- control_p4; // tangenta in punctul final


vec3 Hermite (float u)
{ //întoarce punctul de pe curba
  return (2*u*u*u - 3*u*u +1)*control_p1 + (-2*u*u*u + 3*u*u)*control_p4
          + (u*u*u - 2*u*u + u)*Tp1 + (u*u*u - u*u)*Tp4; }
```

Afisarea curbelor Hermite folosind Geometry shader (5)

```
//Geometry shader (GS) - continuare
void main()
{  mat4 ViewProj =  Projection*View;
   float du = 1.0/float(no_of_generated_points);
   switch (tip_curbag){
case 0: // generare curba Hermite
    for( float u=0; u<1; u+=du)
      { vec3 point = Hermite(u);//se obtin puncte de pe curba in coord. globale
        gl_Position = ViewProj * vec4(point, 1);
        EmitVertex();
      }
  vec3 point = control_p4;//punctul final al curbei
  gl_Position = ViewProj * vec4(point, 1); EmitVertex();
  EndPrimitive(); break;
 case 10:
     // generare vectori tangenti in extremitati
……................................……..
```

Afisarea curbelor Hermite folosind Geometry shader (6)

```
//Fragment shader

layout(location = 0) out vec4 out_color;
uniform int tip_curbag;
void main()
{
out_color = vec4(0,0,0,0);//poligon
if (tip_curbag == 0 || tip_curbag==2)
   out_color = vec4(1,0,0,0);//Hermite sau B-spline2
else
if(tip_curbag==1 || tip_curbag==3)
   out_color = vec4(0,0,1,0);//Bezier sau B-spline 3
}
```

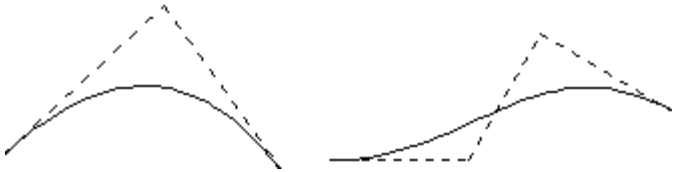
Afisarea curbelor Hermite folosind Geometry shader (7)

// Functia Update

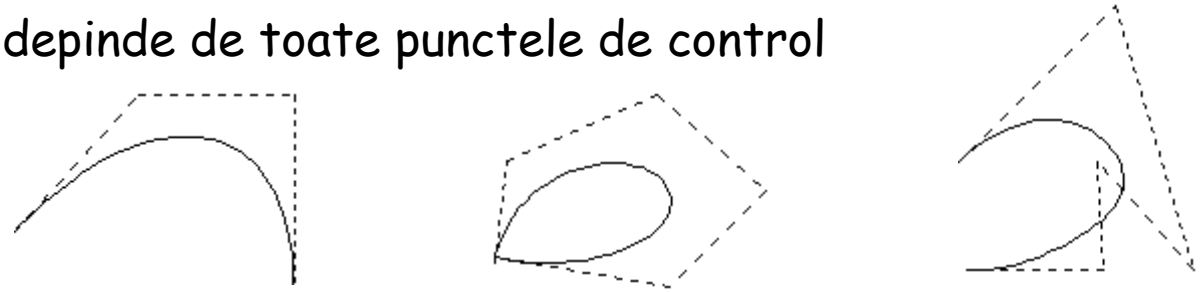
```
Shader *shader = shaders["CurveGeneration"];
shader->Use(); //se activeaza cele 3 shadere folosite la generarea curbelor
Mesh* mesh = meshes["curve"];

glUniform3f(glGetUniformLocation(shader->program, "control_p1"), control_p1.x,
control_p1.y, control_p1.z);// seteaza variabila unforma control_p1
//……..la fel pentru: control_p2, control_p3 si control_p4

glUniform1i(glGetUniformLocation(shader->program, "no_of_generated_points"),
no_of_generated_points); //seteaza variabila unforma no_of_generated_points

if (tip_curba == 0) //Hermite: tipul curbei este setat din tastatura
 { //desenare tangente in punctele extreme
   glUniform1i(glGetUniformLocation(shader->program, "tip_curbag"), 10);
   RenderMesh(mesh, shader, glm::mat4(1));
 // desenare curba
   glUniform1i(glGetUniformLocation(shader->program, "tip_curbag"), tip_curba);
   RenderMesh(mesh, shader, glm::mat4(1));
}
```

Curbe Bezier(1)

- Introduse in 1962, de catre matematicianul Pierre Bézier, pt proiectarea caroseriilor automobilelor la uzinele Renault.
- **Datele geometrice prin care este definita o curba Bézier** : o secventa de puncte, P0, P1,….,Pn – varfurile poligonului de control al curbei, care determina forma si pozitia curbei → curbele Bézier sunt curbe de aproximare
- **Proprietati**
  - Curba trece numai prin primul si ultimul punct din secventa data si este tangenta in primul punct la vectorul P0-P1 si in ultimul punct la vectorul Pn-1 - Pn
  - Gradul polinomului care modeleaza curba depinde de numarul de puncte de control



  - Forma curbei depinde de toate punctele de control



Curbe Bezier(2)

Modelarea matematica

Forma geometrica vectoriala

$p(u) = \Sigma P_i * B_{i,n}(u) \quad 0 <= u <= 1$

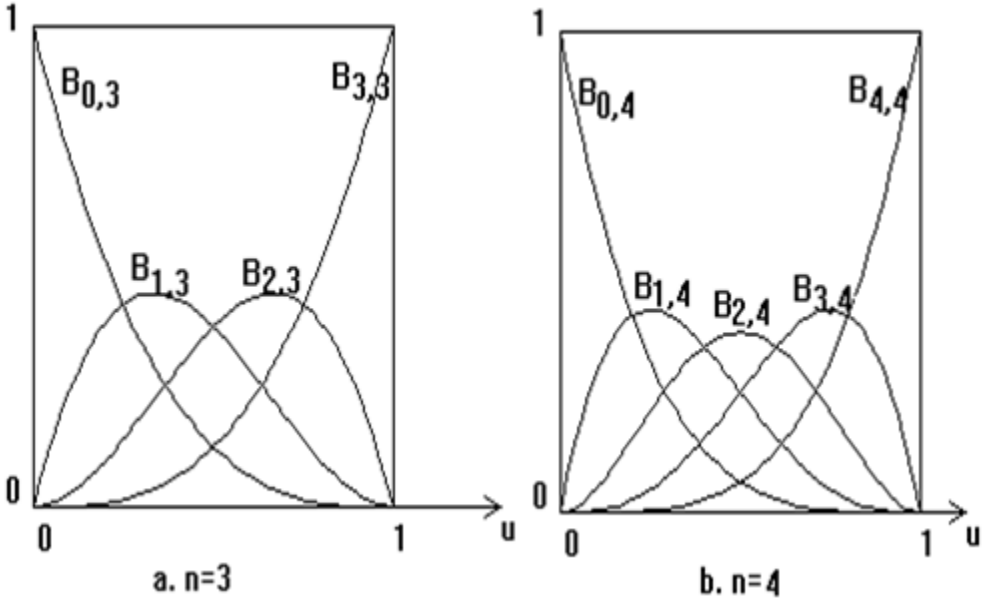
$0 <= i <= n$

Pi – punctele de control

$B_{i,n}(u)$ – polinoamele Bernstein de grad n

$B_{i,n}(u) = (n! * u^i \ (1-u)^{(n-i)}) / (i!*(n-i)!)$

polinoame de grad n pentru (n+1) puncte de control



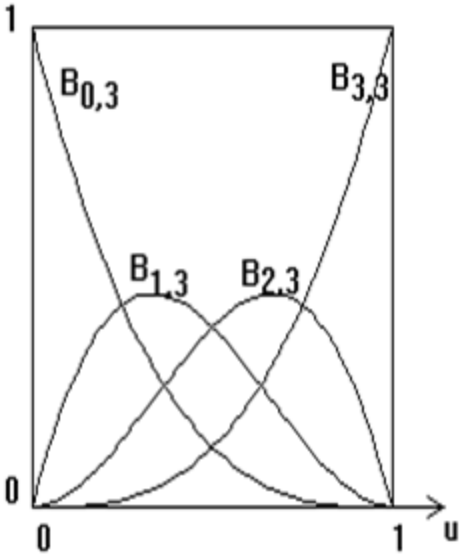
Functiile de ponderare pentru curbele Bezier date prin n+1 puncte de control

Polinoamele Bernstein sunt functiile de ponderare (amestec - blending) pentru curbele Bezier: $p(u) = P_0*B_{0,3}(u) + P_1*B_{1,3}(u) + P_2*B_{2,3}(u) + P_3*B_{3,3}(u)$ //pentru n=3

$B_{i,n}(u)$ sunt nenule pe intreg intervalul de definitie ($0 <= u <= 1$)

- punctele de pe curba depind de toate punctele de control!

Curbe Bezier(3)



Cubica Bezier: 4 puncte de control

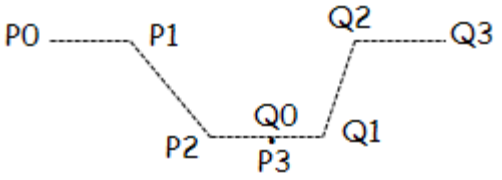
$p(u) = P_0*B_{0,3}(u) + P_1*B_{1,3}(u) + P_2*B_{2,3}(u) + P_3*B_{3,3}(u)$

$p(u) = P_0*(1-u)^3 + P_1*(3*u*(1-u)^2) + P_2*(3*u^2*(1-u)) + P_3*u^3$

Compunerea curbelor Bezier:

De ce este necesara?

- Gradul polinomului de aproximare depinde de numarul de puncte de control
- In practica nu se folosesc pentru modelarea curbelor functii polinomiale cu grad mai mare de 3 → numai 4 puncte de control pentru modelarea printr-o curba Bezier.
- Doua curbe Bezier, C1[P0, P1, P2, P3] si C2[Q0, Q1, Q2, Q3] pot fi compuse dacă este asigurată continuitatea geometrică de ordinul 1 în punctul de jonctiune:

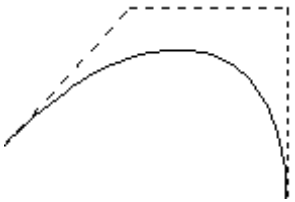


- Q0 coincide cu P3
- Segmentele P2-P3 si Q0-Q1 sunt colineare

Curbe Bezier(4)

Avantaje:

- Usor de definit în mod interactiv

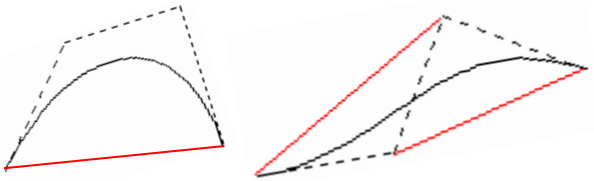


-Poligonul de control permite stabilirea formei si pozitiei curbei

-Curba îşi are extremitatile in primul si ultimul varf al poligonului de control

-Tangentele în punctele extreme sunt primul si ultimul segment al poligonului de control

- Curba este complet inchisa in figura convexă definita de punctele de control (proprietatea de **închidere convexă)**



-**Invarianţa afină**: curba obtinuta prin aplicarea unei transformari afine punctelor sale se poate obtine transformand numai punctele de control si apoi calculand punctele de pe curba.

- Curbele Bezier de grad 2 sunt folosite in proiectarea caracterelor din fontul TrueType.

Dezavantaje:

-Numarul de puncte de control determina gradul polinomului de aproximare

-Functiile de amestec sunt nenule pe intreg intervalul de definitie →fiecare punct de pe curba depinde de toate punctele de control (**control global**)

→ nu putem ajusta local forma curbei

Curbe Bezier(5)

Forma matriciala pentru cubica Bezier:

$$p(u) = P_0 * B_{0,3}(u) + P_1 * B_{1,3}(u) + P_2 * B_{2,3}(u) + P_3 * B_{3,3}(u)$$

$$p(u) = [B_{0,3}(u)\ \ B_{1,3}(u)\ \ B_{2,3}(u)\ \ B_{3,3}(u)] * [P_0\ \ P_1\ \ P_2\ \ P_3]^T$$

$$= [u^3\ \ u^2\ \ u\ \ 1] * M_B * [P_0\ \ P_1\ \ P_2\ \ P_3]^T$$

$$= \boxed{U * M_B * G_B}$$ Forma geometrica matriciala pentru curbele Bezier cubice

$$M_B = \begin{vmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{vmatrix}$$ Matricea de bază Bezier

G_B Vectorul conditiilor geometrice pentru o curba cubica Bezier

Afisarea unei cubice Bézier (1)

// Geometry shader

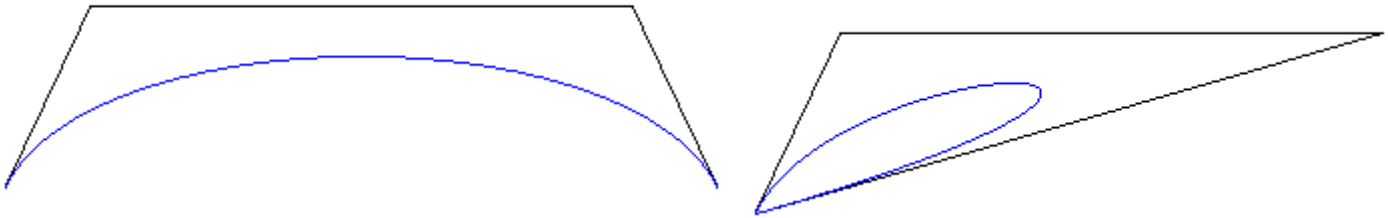
```
layout(lines) in; // nu se folosesc intrari
layout(line_strip, max_vertices = 256) out;

uniform int tip_curbag;
uniform mat4 View, Projection;
uniform vec3 control_p1, control_p2, control_p3, control_p4;
uniform int no_of_generated_points;
```

vec3 Bezier(float u)

{// $p(u) = P_0*(1-u)^3 + P_1*(3*u*(1-u)^2) + P_2*(3*u^2*(1-u)) + P_3*u^3$

return control_p1 * (1 - u)*(1 - u)*(1 - u) + control_p2 * 3 * u*(1 - u)*(1 - u) +

control_p3 * 3 * u*u*(1 - u) + control_p4 *u*u*;}



Afisarea unei cubice Bézier (2)

// Geometry shader - cont

```
void main()
{   mat4 ViewProj =  Projection*View;
    float du = 1.0/float(no_of_generated_points);

    switch (tip_curbag){
    case 1: //genereaza puncte de pe curba Bezier
          for( float u=0; u<1; u+=du)
          {   vec3 bez = Bezier(u);
              gl_Position = ViewProj * vec4(bez, 1);
              EmitVertex();
          }
          vec3 bez = control_p4;
          gl_Position = ViewProj * vec4(bez, 1);
          EmitVertex();
          EndPrimitive();
          break;
    case 11: // generare poligon control Bezier
    ……................................
```

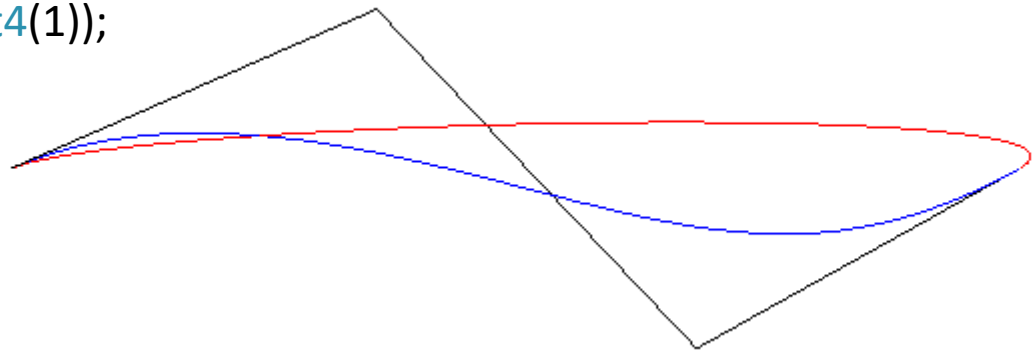
Afisarea unei curbe cubice Bézier (3)

```
// Functia Update
Shader *shader = shaders["CurveGeneration"];
shader->Use(); //se activeaza cele 3 shadere folosite la generarea curbelor
Mesh* mesh = meshes["curve"];

glUniform3f(glGetUniformLocation(shader->program, "control_p1"), control_p1.x, control_p1.y, control_p1.z);
//.............la fel pentru: control_p2, control_p3 si control_p4.................

glUniform1i(glGetUniformLocation(shader->program, "no_of_generated_points"), no_of_generated_points);

if (tip_curba == 1)
{ //desenare poligon control: polilinia care uneste cele 4 puncte de control
  glUniform1i(glGetUniformLocation(shader->program, "tip_curbag"), 11);
  RenderMesh(mesh, shader, glm::mat4(1));
// desenare curba
        glUniform1i(glGetUniformLocation(shader->program, "tip_curbag"), tip_curba);
        RenderMesh(mesh, shader, glm::mat4(1));
}
```

Afisarea Hermite si Bézier (3)
Functia Update

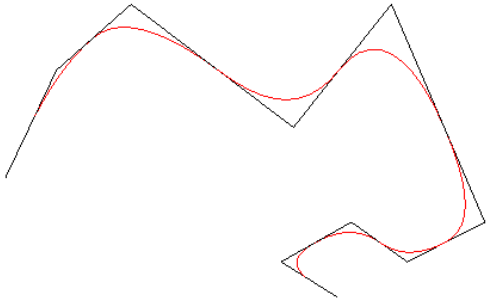
// Functia Update - cont

```
if (tip_curba == 5) //afisare Hermite si Bezier cu aceleasi puncte de control
{ //desenare poligon control
 glUniform1i(glGetUniformLocation(shader->program, "tip_curbag"), 11);
 RenderMesh(mesh, shader, glm::mat4(1));
 //desenare Hermite
glUniform1i(glGetUniformLocation(shader->program, "tip_curbag"), 0);
RenderMesh(mesh, shader, glm::mat4(1));
//desenare Bezier
glUniform1i(glGetUniformLocation(shader->program, "tip_curbag"), 1);
RenderMesh(mesh, shader, glm::mat4(1));
}
```



Curbe B-spline(1)

- **Datele geometrice care definesc o curba B-spline**: o secventa de puncte care determina forma si pozitia curbei – varfurile poligonului de control al curbei.
- Sunt curbe de aproximare



- **Elimina dezavantajele curbelor Bezier:**
  - Gradul polinomului care modeleaza o curba B-spline **nu depinde** de numarul punctelor de control
  - Punctele de pe o curba B-spline nu depind de toate punctele de control
  - ➢ Deplasarea unui punct de control afecteaza forma curbei local (proprietatea de **control local**)

Curbe B-spline(2)

Modelarea matematica

Forma geometrica vectoriala

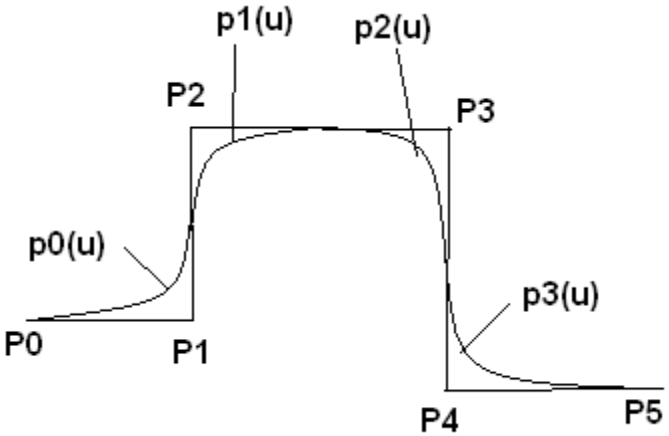
$p(u) = \Sigma P_i * N_{i,k}(u)$ $\quad 0 <= i <= n$

P_i – punctele de control

$N_{i,k}$ – functiile B-spline de grad (k-1), k=3,4

$N_{i,3}$ – polinom de grad 2

$N_{i,4}$ – polinom de grad 3



Curba B-spline de grad 2

Gradul polinoamelor $N_{i,k}$ nu depinde de numarul de puncte de control!

$N_{i,k}$ sunt functii polinomiale definite pe portiuni:

- O curba B-spline se obtine prin jonctiunea mai multor segmente de curba, fiecare segment fiind determinat de:

-3 puncte de control consecutive – B-spline de grad 2

-4 puncte de control consecutive – B-spline de grad 3

Curbe B-spline(3)

Funcţiile B-spline de grad (k-1), k= 3 sau 4

$N_{i,1}(u) = 1$, pentru $t_i <= u <= t_{i+1}$

0, altfel

$$N_{i,k}(u) = \frac{u - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(u) + \frac{t_{i+1} - u}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(u)$$

Fiecare functie $N_{i,k}(u)$ este nenula pe numai k intervale consecutive!

$t_0, t_1, t_2, .., t_n$ – valori nodale: asociaza variabila u punctelor de control

$t_i <= t_{i+1}$

daca $t_{i+1} - t_i = 1$ pentru orice i: functii B-spline uniforme

Curbe B-spline(4)

Vectorul uniform al valorilor nodale pentru o curba B-spline de grad (k-1) deschisa, definita prin n+1 puncte de control:

$t_i = 0$ pentru $i<k$

$t_i = i-k+1$ pentru $k<= i <= n$

$t_i = n-k+2$ pentru $i>n$

$0<= i <= n+k$

$0<= u <= n-k+2$

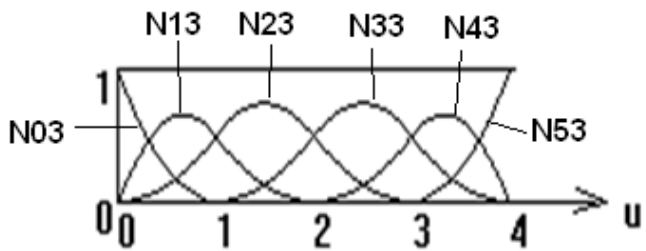
Exemplu:

k=3, n=5 (6 puncte de control) → $0<= i <= 8$, $0<= u <= 4$

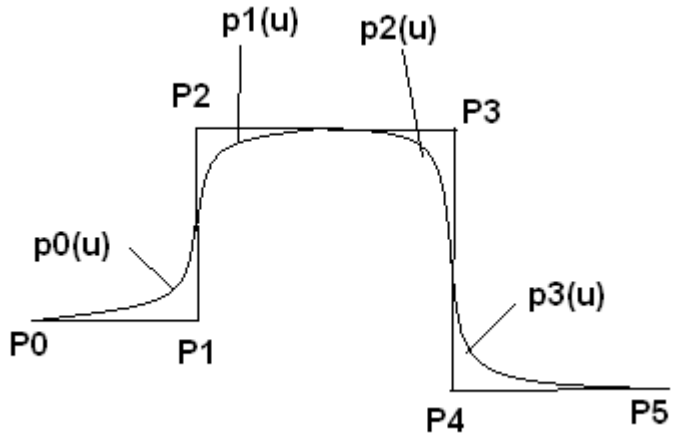
Vectorul valorilor nodale este:

$$\underbrace{t_0 = 0,\ t_1 = 0,\ t_2 = 0}_{i<k},\ \underbrace{t_3 = 1,\ t_4 = 2,\ t_5 = 3}_{k<= i <= n},\ \underbrace{t_6 = 4,\ t_7 = 4,\ t_8 = 4}_{i>n}$$

Curbe B-spline de grad 2, uniforme, neperiodice



Functii B-spline uniforme de grad 2 , neperiodice



Curba B-spline de grad 2 uniforma neperiodica

0<=u<1 p(u) = P0*N03(u) + P1*N13(u) + P2*N23(u)
1<=u<2 p(u) = P1*N13(u) + P2*N23(u) + P3*N33(u)
2<=u<3 p(u) = P2*N23(u) + P3*N33(u) + P4*N43(u)
3<=u<4 p(u) = P3*N33(u) + P4*N43(u) + P5*N53(u)

Ecuatia curbei

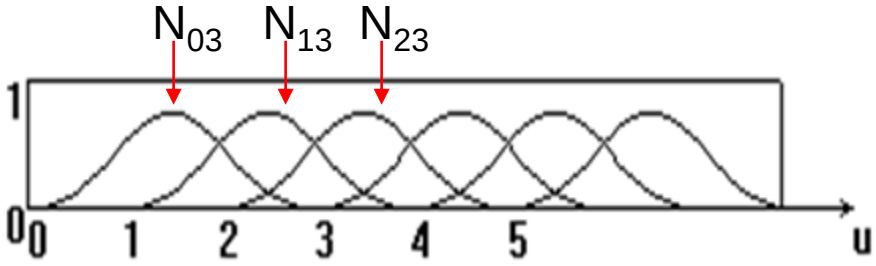
$$p(u) = \Sigma P_i * N_{i,k}(u) \qquad 0 <= i <= n$$

i<=u<i+1 pi(u) = ΣPj*Nj3(u) - ecuatia segmentului de curba pe intervalul [i, i+1)
i<= j <= i+2

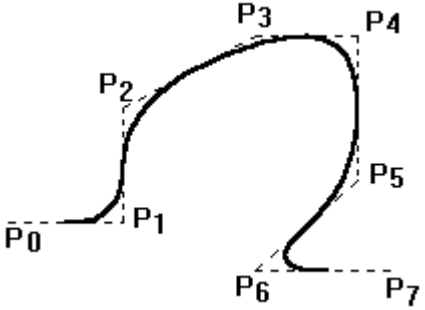
- Curba B-spline definita prin 6 puncte de control este compusa din 4 segmente de curba conectate cu continuitate geometrica de ordinul 1 in punctele de jonctiune.
- Prin adaugarea unui punct de control se adauga un nou segment la curba

Curbe B-spline de grad 2, uniforme periodice

- Functiile B-spline neperiodice au aceeasi forma numai in interiorul intervalului pe care sunt definite!
- Functiile B-spline uniforme periodice au aceeasi forma pe orice interval $i <= u < i+1$, fiind nenule pe numai:
  - -3 intervale consecutive (grad 2)
  - -4 intervale consecutive (grad 3)



Functiile b-spline uniforme periodice de grad 2



Curba B-spline de grad 2 uniforma periodica

Curbe B-spline de grad 2 uniforme periodice

- Deoarece funcţiile B-spline uniforme periodice au aceeasi forma pe orice interval (i→i+1), se poate utiliza o forma normalizata a ec.

Parametrice: $p_i(u) = \Sigma P_j * N_{j,3}(u), \quad i <= j <= i+2$

$0 <= u < 1$

Ecuatia unui segm. de curba B-spline 2 uniforma periodica de grad 2

sau $p_i(u) = N_{i-1,3}(u) * P_{i-1} + N_{i,3}(u) * P_i + N_{i+1,3}(u) * P_{i+1}$

$p_i(u) = 0.5*[(1-u^2)*P_{i-1} + (-2*u^2 + 2*u + 1)*P_i + u^2*P_{i+1}]$ $\quad 1 <= i <= n-2,\ 0 <= u < 1$

$p_i(u) = [\ F1(u)\ \ F2(u)\ \ F3(u)\ ][\ P_{i-1}\ \ P_i\ \ P_{i+1}]^T$

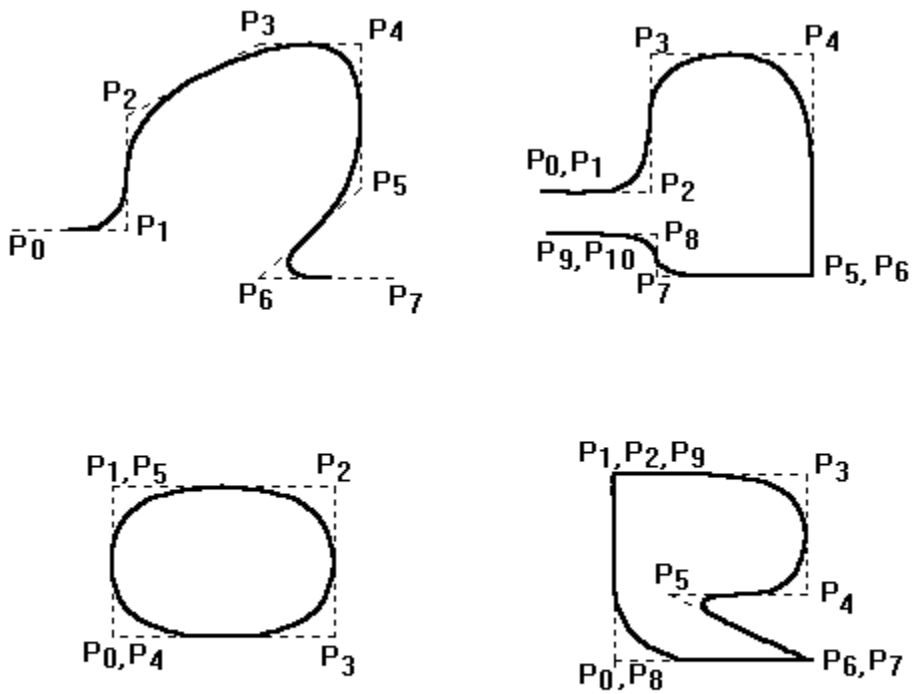
$p_i(u) = [u^2\ \ u\ \ 1]\ Ms3\ [\ P_{i-1}\ \ P_i\ \ P_{i+1}]^T$

Ms3 – matricea de baza B-spline de grad 2

$$Ms3 = 0.5* \begin{vmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 2 & 1 & 0 \end{vmatrix}$$

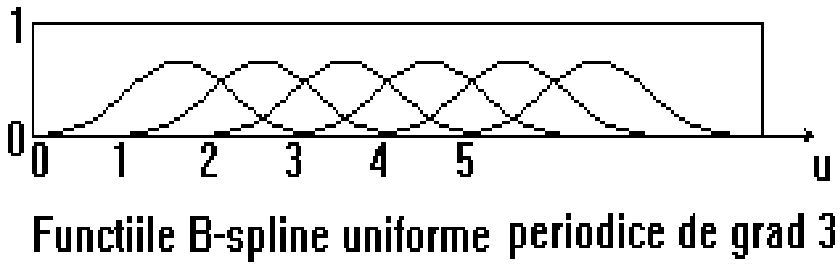
$\boxed{p_i(u) = U2*Ms3*Gs3_i}$ – forma geometrica matriciala a unui segment de curba B-spline uniformă periodică de grad 2

Curbe B-spline de grad 2 uniforme periodice



Curbe B-spline uniforme periodice de grad 2.

Curbe B-spline uniforme periodice de grad 3



Functiile B-spline uniforme periodice de grad 3

Ecuatia unui segment de curba B-spline uniforma periodica de grad 3:

$$p_i(u) = [u^3\ u^2\ u\ 1]\ Ms4\ [P_{i-1}\ \ P_i\ \ P_{i+1}\ \ P_{i+2}]^T \qquad 1<=i<=n-3,\ \ 0<=u<1$$

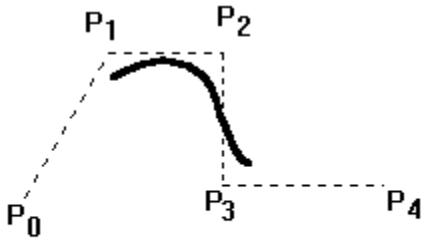
Ms4 – matricea de baza B-spline de grad 3

$$Ms4 = (1/6)^* \begin{vmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{vmatrix}$$

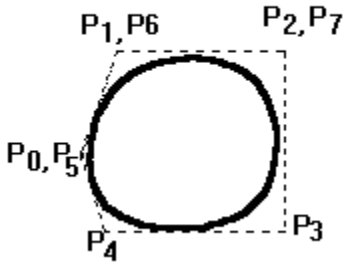
$$\boxed{p_i(u) = U*Ms4*Gs4_i}$$

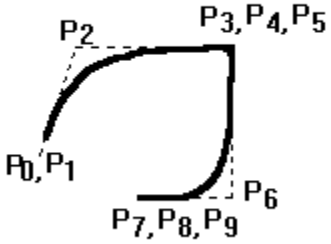
– forma geometrica matriciala a unui segment de curba B-spline uniformă periodică de grad 3

Curbe B-spline uniforme periodice de grad 3



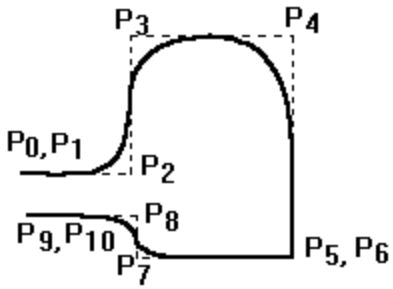
Curba B-spline de grad 3 netezeste poligonul de control mai mult decat curba B-spline de grad 2!

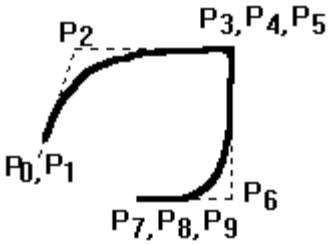




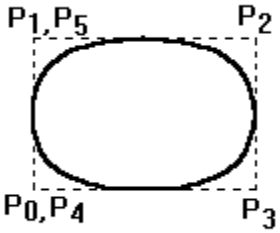
Proprietatile curbelor B-spline uniforme periodice

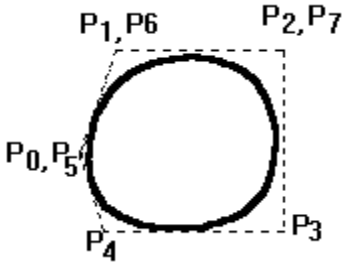
-Puncte de control confundate





-Curbe inchise

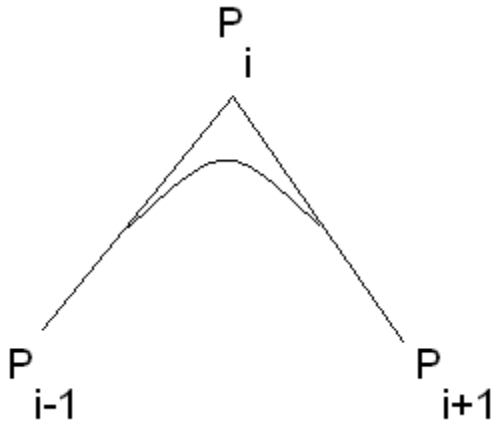




-Inchidere convexa

-Invarianta afina

Proprietati ale curbelor B-spline uniforme periodice de grad 2



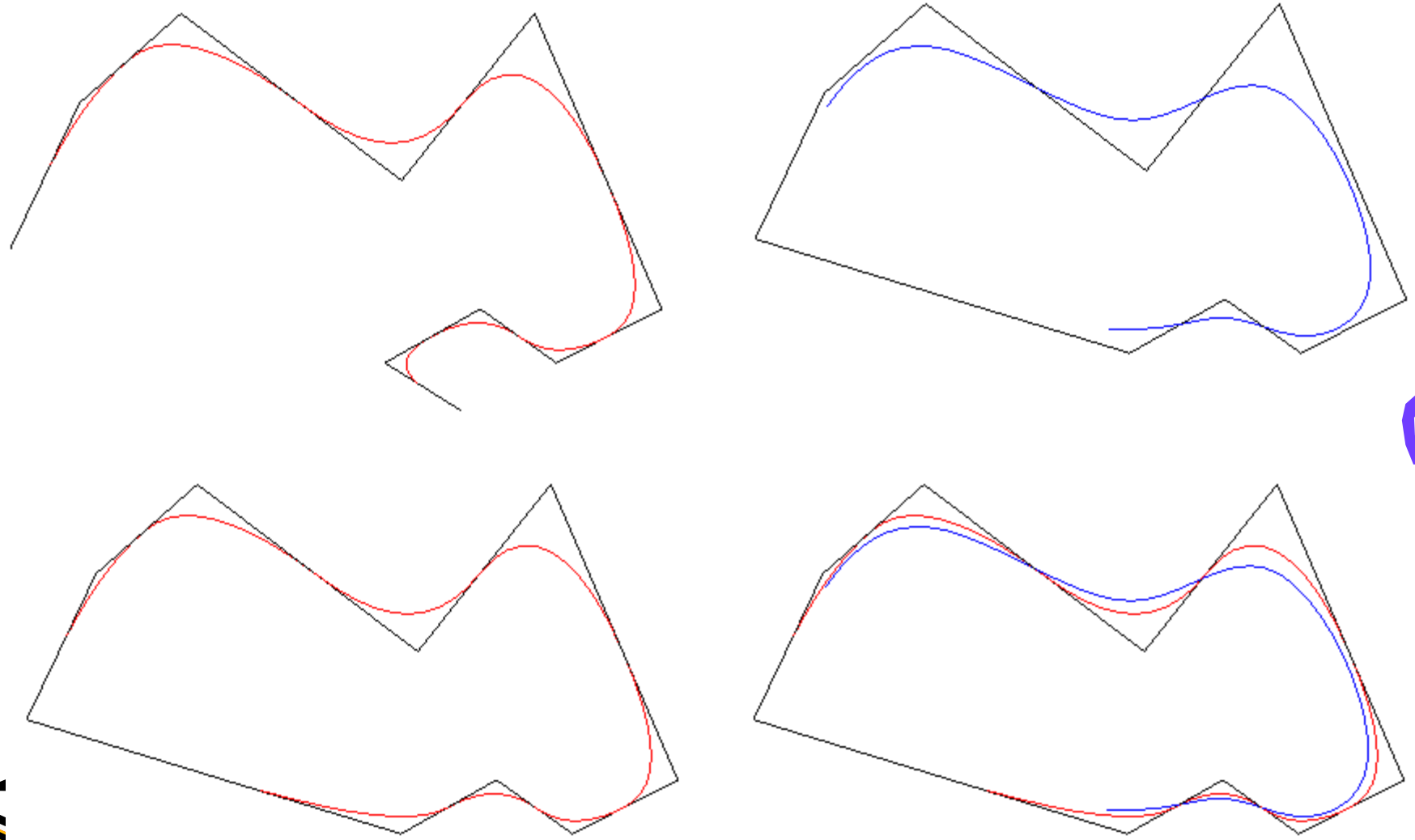
Afisarea unei curbe B-spline (1)
Geometry shader

```
layout(lines) in;
layout(line_strip, max_vertices = 256) out;

uniform mat4 View;
uniform mat4 Projection;
uniform int tip_curbag;

uniform vec3 control_points[20];
uniform int num_control_points;
uniform int no_of_generated_points;//nr. de puncte generate pt fiecare segment al curbei B-spline

vec3 Bspline2(float t, int i) { //întoarce un punct de pe segmentul de curba i
            return (0.5*(1-2*t + t*t)*control_points[i-1] + 0.5*(-2*t*t + 2*t +1)*control_points[i]
 + 0.5*t*t*control_points[i+1]);
}
vec3 Bspline3(float t, int i) {//întoarce un punct de pe segmentul de curba i
   return  (1.0/6*(-t*t*t + 3*t*t - 3*t +1)*control_points[i-1] +
            1.0/6*(3*t*t*t-6*t*t+4)*control_points[i] +
            1.0/6*(- 3*t*t*t + 3*t*t + 3*t +1)*control_points[i+1] + 1.0/6*t*t*t*control_points[i+2]);
 }
```

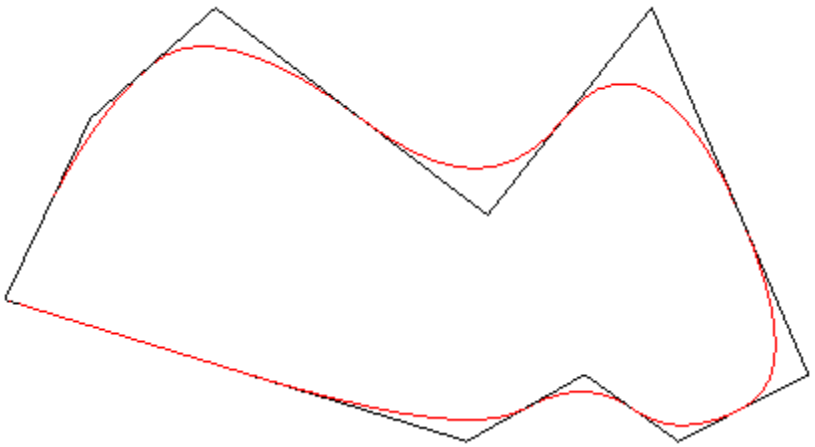
Afisarea unei curbe B-spline (2)
Geometry shader

```
void main()
{ mat4 ViewProj =  Projection*View;
  float du = 1.0/float(no_of_generated_points);
   switch (tip_curbag){
      case 3: // curba B-spline de grad 2
         for(int i=1; i<=num_control_points-2; i++) //genereaza toate segm de curba
            //se genereaza segmentul de curba i
             for( float u=0; u<1; u+=du)
             {vec3 bspl = Bspline2(u,i); gl_Position = ViewProj * vec4(bspl, 1); EmitVertex();
             }
        EndPrimitive(); // s-a generat toata curba
        break;
     case 4: // curba B-spline de grad 3
        for(int i=1; i<num_control_points-3;i++)
          //se genereaza segmentul de curba i
          for( float u=0; u<1; u+=du)
            {vec3 bspl = Bspline3(u,i); gl_Position = ViewProj * vec4(bspl, 1); EmitVertex();
            }
       EndPrimitive();
       break;
   }
}
```

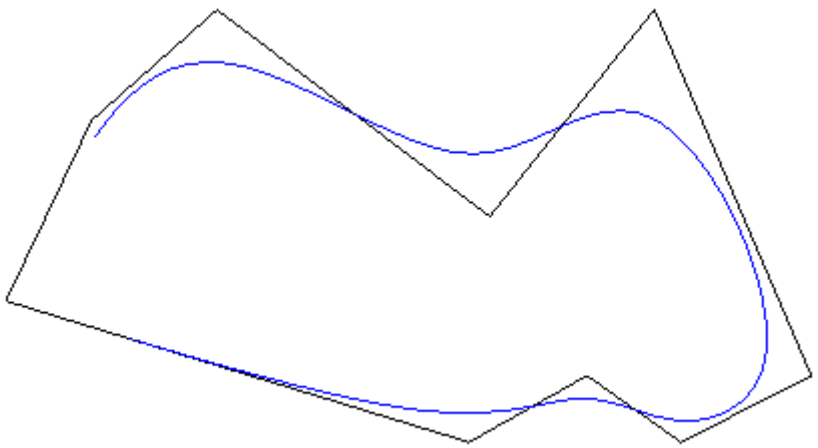
Exemple de curbe B-spline(1)
Roşu – grad 2; albastru – grad 3

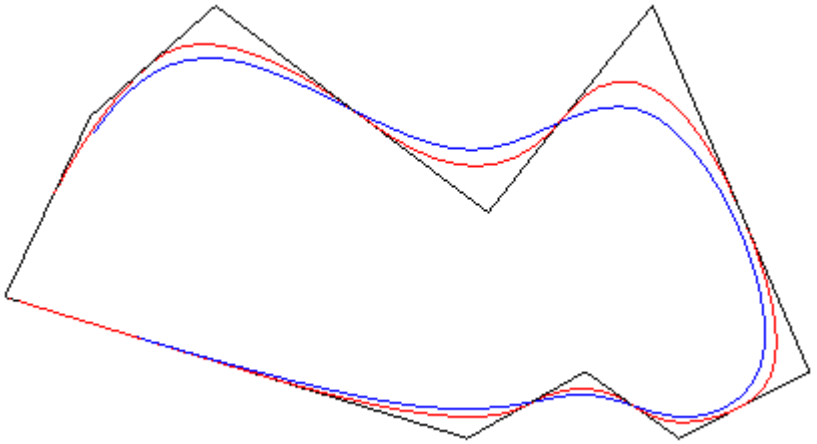


Exemple de curbe B-spline(2)



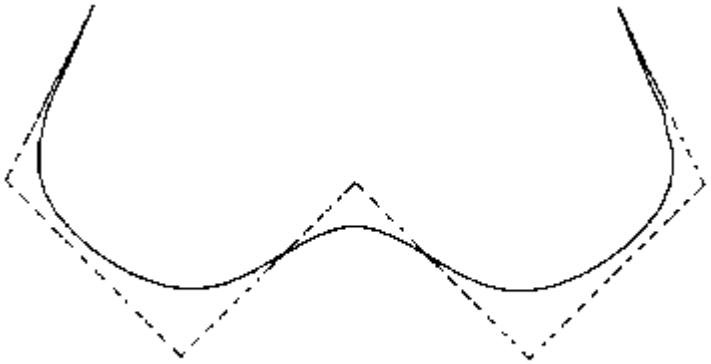
Ultimele 2 puncte de control coincid cu primul.





Curbe Nurbs(1): curbe B-spline generalizate

- **Sunt curbe B-spline neuniforme şi raţionale: Non-Uniform Rational B-Spline**
- **Datele geometrice care definesc o curba Nurbs:**
  - Secvenţa de puncte care determina forma şi poziţia curbei
  - Ponderile asociate punctelor de control – prin care poate fi controlată forma curbei



Curba NURBS definita prin 7 puncte de control

Curbe Nurbs(2)

Curbe B-spline ne-uniforme

$p(u) = \sum P_i * N_{i,k}(u) \quad 0<=i<=n,\ umin<=u<=umax$

P_i - punctele de control
$N_{i,k}(u)$ – functiile B-spline de grad (k-1)

Vectorul valorilor nodale folosit in definitia functiilor B-spline, t_o, t_1, t_2, …. ,t_n, care asociaza variabila u punctelor de control, **este neuniform**: diferenta ($t_{i+1} – t_i$), k+1<=i <= n-1, nu este constanta.

$$t: \underbrace{a, a, a, \ldots, a}_{k}, t_{k+1}, \ldots, t_n, \underbrace{b, b, b, \ldots, b}_{k}$$

Curbe Nurbs(3)

Curbe B-spline rationale

Reprezentarea unui punct de control, $P_i(x_i,y_i,z_i)$ in coordonate omogene este:

$C_i = (x_i*w_i,\ y_i*w_i,\ z_i*w_i,\ w_i)$

Pentru punctele de control ale unei curbe B-spline ne-rationala $w_i=1$

→ $Ci = (x_i,\ y_i,\ z_i,\ 1)$

→Aplicarea unei transformari perspectiva asupra unui punct de control $(x_i,\ y_i,\ z_i,\ 1)$ poate produce un punct pentru care coordonata omogena este diferita de 1

→Curbele B-spline ne-rationale sunt invariante numai la transformari afine, nu si la transformari perspectiva.

Curbe Nurbs(4)

Ecuatia unei curbe NURBS în coordonate omogene este:

$p_w(u) = \sum N_{i,k}(u)*Ci$, umin <= u <= umax, 0<=i<=n

$C_i = (x_i*w_i, y_i*w_i, z_i*w_i, w_i)$

Pe componente:

$x_w(u) = \sum N_{i,k}(u)*x_i*w_i$

$y_w(u) = \sum N_{i,k}(u)*y_i*w_i$

$z_w(u) = \sum N_{i,k}(u)*z_i*w_i$

$w(u) = \sum N_{i,k}(u)*w_i$

Conversia din coordonate omogene in coordonate carteziene:

$x(u) = x_w(u)/w(u)$

$y(u) = y_w(u)/w(u)$

$z(u) = z_w(u)/w(u)$

Ecuatia curbei in coordonate carteziene:

$p(u) = (\sum N_{i,k}(u)*P_i*w_i)/(\sum N_{i,k}(u)*w_i)$

umin <= u <= umax, 0<=i<=n

Curbe Nurbs(5)

Valorile w_i asociate punctelor de control se numesc ponderi:

- Cand w_i creste, curba se apropie de punctul de control
- Cand w_i scade, curba se indeparteaza de punctul de control
- Daca w_i= 0, punctul de control nu are efect asupra curbei

Ecuatia curbei Nurbs in coordonate carteziene:

$p(u) = (\sum N_{i,k}(u)*P_i*w_i)/(\sum N_{i,k}(u)*w_i)$ umin <= u <= umax, 0<=i<=n

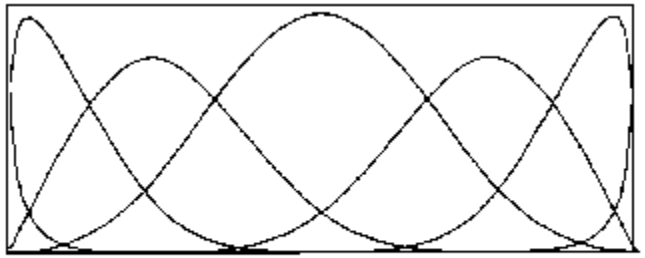
se mai poate scrie:

$p(u) = \sum R_{i,k}(u)*P_i$ $R_{i,k}(u)$ se numesc functii B-spline neuniforme, rationale

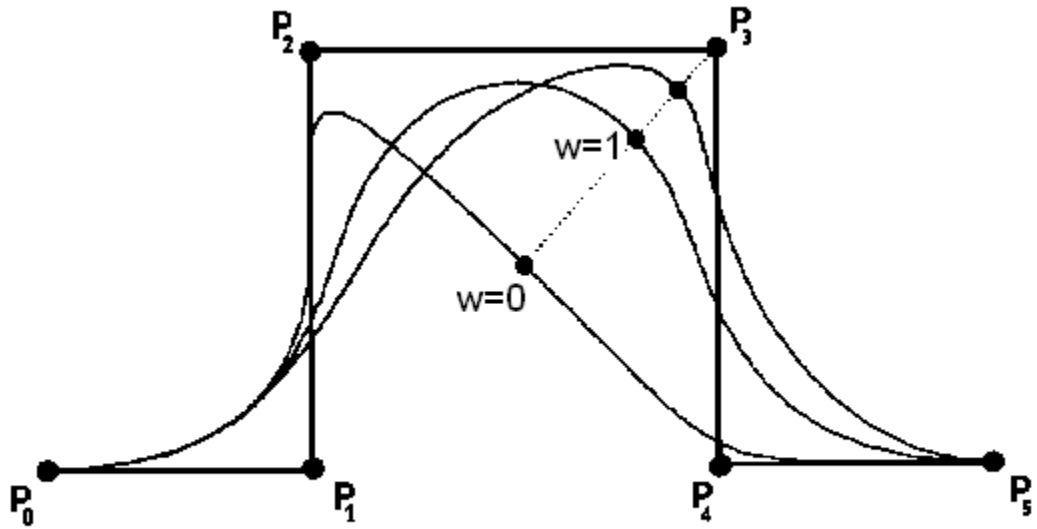
$$R_{i,k}(u) = \frac{w_i * N_{i,k}(u)}{\sum N_{j,k}(u)*w_j \quad 0<= j <=n} \qquad 0<= i <= n$$

Pentru w_i =1, 0<=i<=n, $R_{i,k}(u) = N_{i,k}(u)$ deoarece $\sum N_{i,k}(u)=1$

Curbe Nurbs(3)



Functiile de amestec pentru curbele NURBS de grad 3: $\mathbf{R_{i,4}(u)}$

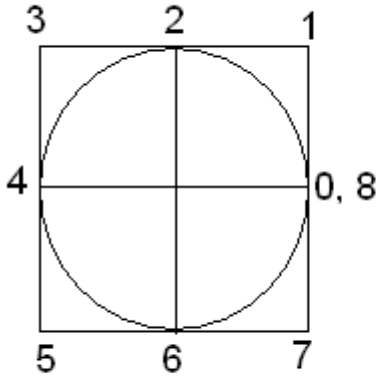


Influenta ponderilor asociate punctelor de control asupra formei curbei

Curbe Nurbs(4)

Proprietati care le confera popularitate(1)

- Ofera o forma matematica comuna pentru curbele analitice standard (de ex. curbe conice) care nu pot fi bine definite prin curbe B-spline şi curbele de forma libera: proprietate utilă în sistemele CAD/CAM.



P_i	x	y	w_i
0	1	0	1
1	1	1	(v2)/2
2	0	1	1
3	-1	1	(v2)/2
4	-1	0	1
5	-1	-1	(v2)/2
6	0	-1	1
7	1	-1	(v2)/2
8	1	0	1

Definitia unui cerc printr-o curba NURBS de grad 2

Curbe Nurbs(5)

Proprietati care le confera popularitate(2)

-Ofera flexibilitate in proiectarea unei game largi de forme → sunt larg utilizate in sistemele CAD/CAM

- Pot fi evaluate destul de rapid prin algoritmi numerici stabili si exacti

-Sunt invariante fata de transformarile afine

-Sunt invariante fata de transformarea perspectiva

CURBE DE FORMA LIBERA - 3

Prof. univ. dr. ing. Florica Moldoveanu

Curbe spline cubice de interpolare

Curba spline cubica de interpolare:

- Definită printr-o secvență de puncte de control, P_0, P_1, P_2,.....P_{n-1}
- Trece prin toate punctele de control date
- Este modelata prin polinoame de grad 3 definite pe portiuni.

Datele geometrice:

- Secventa de puncte de control prin care trebuie sa treaca curba: P_0, P_1, P_2,.....P_{n-1}
- Alte conditii impuse curbei (în functie de tipul curbei):

  - tangenta la curba în fiecare punct de control: P'_0, P'_1, P'_2,.....P'_{n-1}

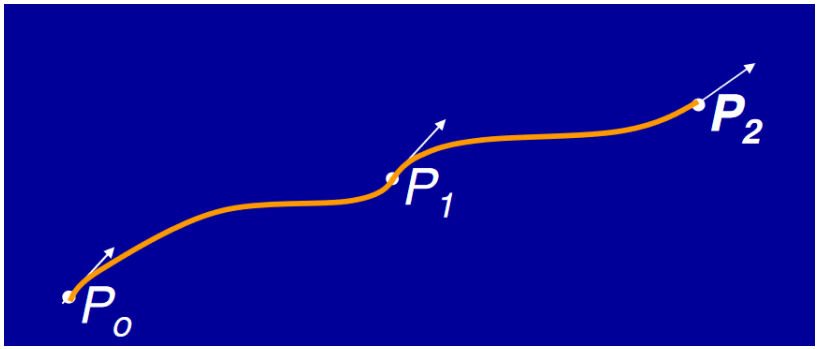
  - curbura în fiecare punct de control: P''_0, P''_1, P''_2,.....P''_{n-1}

- **Curba se obtine prin jonctiunea segmentelor de curbă delimitate de punctele de control**:

  - Fiecare segment de curba este descris printr-un polinom de grad 3
  - Compunerea segmentelor este determinata de conditiile impuse în punctele de jonctiune

Curbe spline cubice Hermite (1)

- O curbă spline Hermite cubica este definită prin:
  - O secvență de n puncte de control, P_0,.., P_k, P_{k+1} .., P_{n-1}
  - Tangentele în cele n puncte



- Fiecare segment de curbă este definit prin 2 puncte de control succesive si tangentele în punctele respective: fiecare segment de curba este o curbă Hermite.
- Ecuatia segmentului de curba determinat de $G_k = [P_k, P_{k+1}, P_k^u, P_{k+1}^u]$ este:

$$p_k(u) = \Sigma a_{i,k} * u^i \quad 0<=i<=3,\ u_k<=u<=u_{k+1}\ ,\ 0<=k<=n-2$$

Curbe spline cubice Hermite (2)

- Considerand toate intervalele $u_k - u_{k+1} = 1$ ($0<=u<=1$) si impunand conditiile pe frontiera obtinem ecuatia matriciala geometrica:

$$p_k(u) = [F1(u)\ \ F2(u)\ \ F3(u)\ \ F4(u)]\ [P_k, P_{k+1}, P_k^u, P_{k+1}^u]^T \quad 0<=u<=1$$

F1, F2, F3, F4 sunt functiile de amestec pentru curbele Hermite.

$F1(u)= 2u^3 - 3u^2 +1$
$F2(u)= -2u^3 + 3u^2$
$F3(u)= u^3 - 2u^2 +u$
$F4(u)= u^3 - u^2$

$$p_k(u) = [u^3\ \ u^2\ \ u\ \ 1] \begin{vmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{vmatrix} [P_k, P_{k+1}, P_k^u, P_{k+1}^u]^T$$

$p_k(u) = U*M_H*G_k$, este ec. matricala a unui segment de curbă unde:

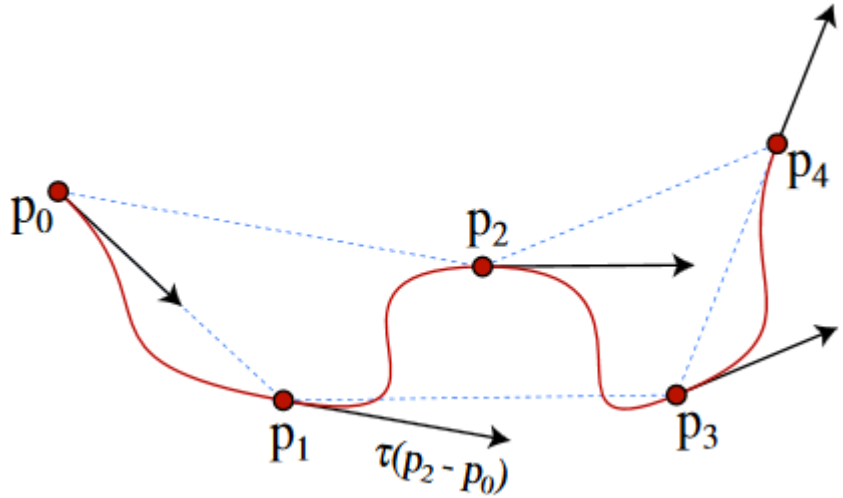
M_H este matricea de baza Hermite

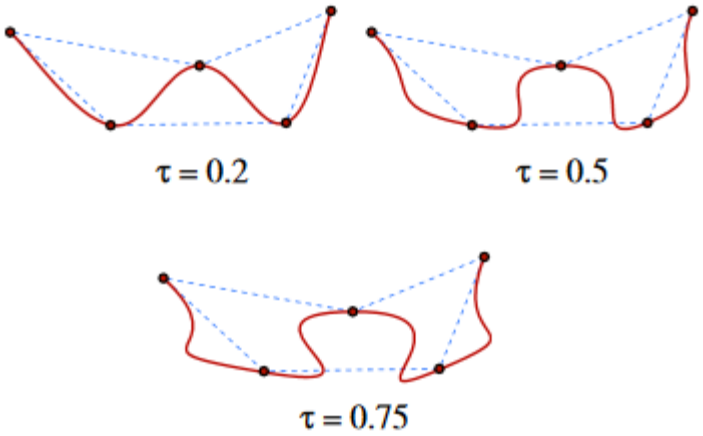
G_k - vectorul cond. geometrice pentru curba Hermite P_k - P_{k+1}

Curba se obtine prin compunerea curbelor Hermite determinate de 2 puncte de control succesive, fiind asigurata continuitatea geometrica de ordin 1 in punctele de jonctiune.

Curbe spline Catmull-Rom (1)

- **Dezavantajul curbelor spline Hermite**: proiectantul trebuie sa defineasca tangente în punctele de control.
- Curbele spline Catmull-Rom sunt definite numai prin punctele de control: tangenta in fiecare punct intermediar P_k este determinata de punctul anterior P_{k-1} si cel urmator P_{k+1}: vectorul $P_{k-1} \rightarrow P_{k+1}$ determină direcția tangentei in P_k





Parametrul τ este numit "tension"; el determina cat de mult se "îndoaie" curba în punctele de control; de regula este setat la 0.5

Curbe spline Catmull-Rom (2)

- Deoarece este o curba cubica, ec. algebrica a unui segment de curba Catmull-Rom este:

  $p_i(u) = \Sigma a_{k,i} * u^k \quad 0<=k<=3,\ 0<= u <=1$

- Segmentul cuprins intre P_{i-1} şi P_i este determinat de punctele de control P_{i-2}, P_{i-1}, P_i, P_{i+1}

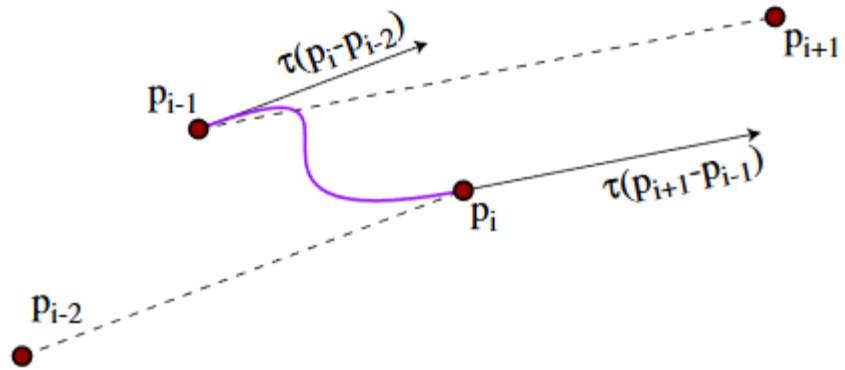
- Impunand conditiile:

$p_i(0) = P_{i-1}$

$p_i(1) = P_i$

$p'_i(0) = \tau(P_i - P_{i-2})$

$p'_i(1) = \tau(P_{i+1} - P_{i-1})$



se obtin coeficientii $a_{0,i}$, $a_{1,i}$, $a_{2,i}$, $a_{3,i}$ (la fel ca pentru curbele Hermite).

- Se introduc coeficientii în ec. algebrica, si se grupeaza termenii dupa conditiile geometrice.

- Se obtine forma geometrica: $p_i(u) = [C1(u)\ C2(u)\ C3(u)\ C4(u)]\ [P_{i-2}, P_{i-1}, P_i, P_{i+1}]^T$

  unde C1(u) C2(u) C3(u) C4(u) sunt functiile de amestec pentru un segment de curba Catmul-Rom.

Curbe spline Catmull-Rom (3)

Ecuatia matriciala a unui segment de curba cubica Catmull-Rom:

$$p_i(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -\tau & 2-\tau & \tau-2 & \tau \\ 2\tau & \tau-3 & 3-2\tau & -\tau \\ -\tau & 0 & \tau & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} [P_{i-2}, P_{i-1}, P_i, P_{i+1}]^T$$

$0<=u<=1$

Matricea de baza Catmull-Rom

Proprietati:

- Trece prin toate punctele de control si are control local.
- Este relativ usor de calculat – tangentele se determina automat.
- Are continuitate geometrica de ordinul 1 in punctele de jonctiune (nu sunt discontinuitati in directia tangentei)
  - Nu are continuitate geometrica de ordinul 2 (derivata de ordinul 2 este interpolata liniar pe fiecare segment)
  - Nu are proprietatea de închidere convexa.

Bibliografie pentru capitolul "Curbe de forma libera"

1. Florica Moldoveanu, Z. Racoviţă, G. Hera, S. Petrescu, M. Zaharia, *Grafica pe Calculator*, Ed. Teora, Bucureşti 1996.

2. Michael E. Mortenson, *Geometric Modeling*, Ed. John Wiley & Sons, 1985.

3. David Solomon, *Curves and Surfaces for Computer Graphics*, Springer, 2006, ISBN -10: 0-387-24196.

4. http://www.cs.cmu.edu/afs/cs/academic/class/15462-s10/www/lec-slides/lec06.pdf

5. *An Interactive Introduction to Splines*, http://ibiblio.org/e-notes/Splines/Intro.htm

6. http://fab.cba.mit.edu/classes/S62.12/docs/Piegl_NURBS.pdf

Suprafete de baleiere spatiala

Prof. univ. dr. ing. Florica Moldoveanu

Curs *Sisteme de Prelucrare Grafică* – UPB, Automatică şi Calculatoare
2021-2022

Modelarea matematica

Reprezentarea analitica a unei suprafete 3D

Forma explicita: P(x, y, z) = [x y z] = [x y f(x,y)]

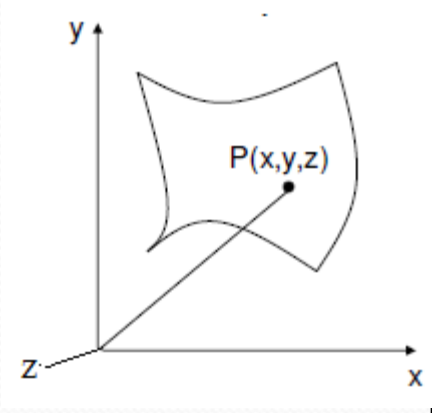
$$f(x,y)=\sum_{m=0}^{p}\sum_{n=0}^{q}a_{mn}x^m y^n$$

P este vectorul de pozitie al unui punct de pe suprafata.

Rar folosita in Computer Graphics.

Forma implicita: f(x,y,z) = 0

Folosita pentru a reprezenta suprafete cunoscute prin ecuatiile lor din geometria analitica (ex. planul, sfera)

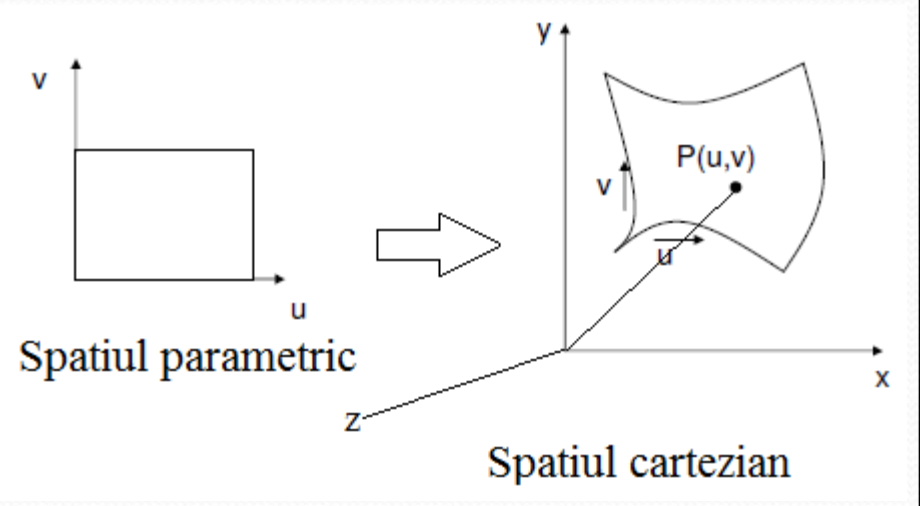


Reprezentarea parametrică

p(u, v) = [x y z] = [f_x(u,v), f_y(u,v), f_z(u,v)]

umin<= u <=umax
vmin<= v <=vmax

Avantaje: - aplicarea transformarilor
- exprimarea conditiilor geometrice care definesc suprafetele
- ec. parametrice pot reprezentata orice suprafata 3D



Suprafete 3D

2 tipuri de suprafete larg utilizate pentru modelarea suprafetelor complexe reale sau virtuale, în sistemele CAD/CAM, în arhitectura, industria de automobile, avioane, proiectarea de obiecte 3D de arta, etc**:**

- **Suprafețele de "baleiere spatiala"(Sweep surfaces)**: obtinute prin rotația/ translația în spațiul 3D a unei drepte sau curbe.

- **Suprafețele de formă liberă (Freeform surfaces):** definite prin puncte din spatiul 3D si alte conditii geometrice atasate punctelor, sau folosind curbe de forma libera.

Suprafete de baleiere spatiala

C(u) – o curba in spatiu, numita "**curba profil**" (sau obiectul generator)

T(w) – o matrice de transformare a curbei C(u), care defineste **regula de "baleiere"** a spatiului

T(w) - include: rotatie sau/si translatie

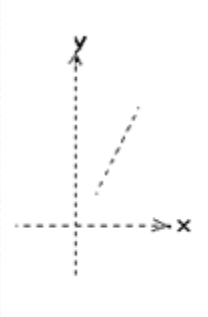
- poate sa includa scalare si forfecare

T(w): numai rotatie → suprafata de rotatie

T(w): numai translatie →suprafata de translatie

Suprafete de rotatie(1)

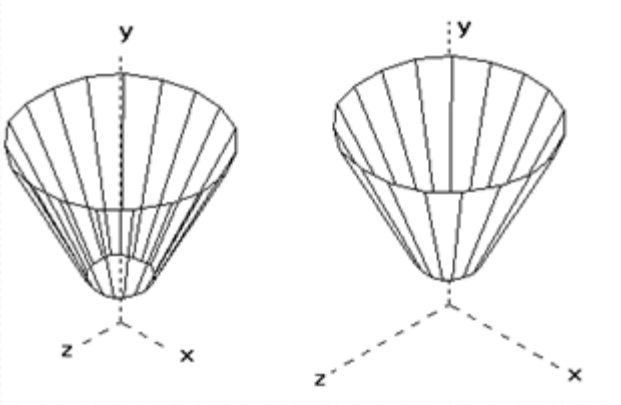
- Se obtin prin rotatia unei curbe C(u) in jurul unei axe 3D



Ec. unui segment de dreapta

p(u) = P1 + u(P2-P1) $u \in [0,1]$

P1, P2 – puncte din planul XOY



Ecuatia vectoriala a segmentului, in coordonate omogene:

$p(u)= [x(u) \quad y(u) \quad z(u) \quad 1]=$

$[\, x_1+u(x_2-x_1) \;\; y_1+u(y_2-y_1) \;\; 0 \;\; 1 \,]$, cu $u \in [0,1]$

$$R(\varphi) = \begin{bmatrix} \cos\varphi & 0 & -\sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ec. suprafetei de rotatie

$S(u, \varphi) = p(u)*R(\varphi) =$

$[\, (x_1+u(x_2-x_1)*\cos(\varphi)) \;\; (y_1+u(y_2-y_1)) \;\; -(x_1+u(x_2-x_1)*\sin(\varphi)) \;\; 1 \,]= [\, x(u, \varphi) \; y(u, \varphi) \; z(u, \varphi) \; 1]$

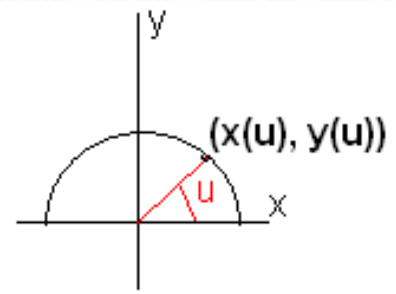
cu $u \in [0,1]$, $\varphi \in [0, 2\Pi]$

Suprafete de rotatie(2)

Ecuatia parametrica a semicercului de raza r centrat in origine

$x(u) = r*\cos(u)$
$y(u) = r*\sin(u)$
$z(u) = o$

$0 <= u <= \pi$



$$R_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotatia in jurul axei OX

$p(u) = [x(u) \quad y(u) \quad z(u) \quad 1]$

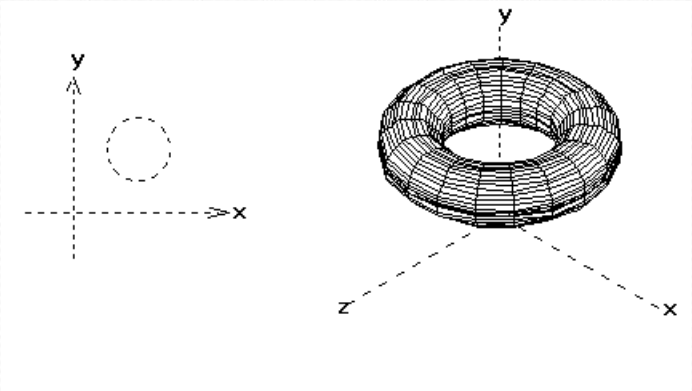
Ecuatia parametrica a sferei obtinute prin rotatia semicercului in jurul axei OX:

$$S(u, \varphi) = p(u)*R_x(\varphi) = [\, r*\cos(u) \quad r*\sin(u)*\cos(\varphi) \quad r*\sin(u)*\sin(\varphi) \quad 1 \,]$$

cu $u \in [0, \pi]$ si $\varphi \in (0, 2\pi)$

$x(u, \varphi) = r*\cos(u)$
$y(u, \varphi) = r*\sin(u)*\cos(\varphi)$ ***Forma pe componente***
$z(u, \varphi) = r*\sin(u)*\sin(\varphi)$

Suprafete de rotatie(3)



$$R(\varphi) = \begin{bmatrix} \cos\varphi & 0 & -\sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ec. cercului

$x(u) = x_c + r*\cos(u)$
$y(u) = y_c + r*\sin(u)$
$z(u) = 0$

$u \in [0,2\pi]$

Ec. torului

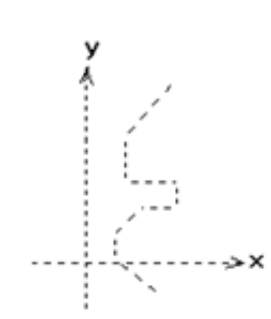
$S(u, \varphi) = p(u)*R(\varphi) = [\, x(u)\ y(u)\ 0\ 1] * R(\varphi) =$

$= [\, (xc+r*\cos(u))*\cos(\varphi)\ \ (yc+r*\sin(u))\ \ -(xc+r*\cos(u)*\sin(\varphi))\ \ 1\,]$

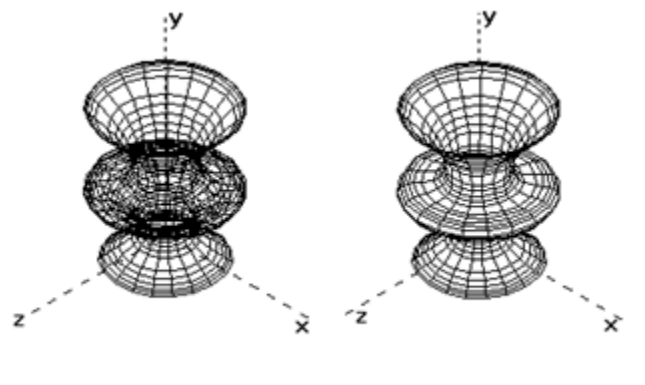
cu $u \in [0,2\pi]$ si $\varphi \in [0,2\pi]$

Suprafete de rotatie(4)

Rotatia unei curbe de forma libera



$$R(\varphi) = \begin{bmatrix} \cos\varphi & 0 & -\sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Ec. matriciala a Cubicei Bezier

$p(u) = U^* M_B * [P0\ P1\ P2\ P3]^T$, $0 <= u <= 1$

Po ,P1 ,P2 ,P3 sunt cele 4 puncte de control

M_B este matricea de baza Bezier

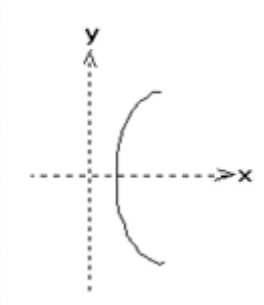
$S_B = p(u) * R(\varphi)$

$$S_B(u,\varphi) = U \cdot M_B \cdot \begin{bmatrix} x_0 & y_0 & z_0 & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{bmatrix} \cdot R(\varphi)$$

$0 <= u <= 1,\ \varphi \in [0,2\pi]$

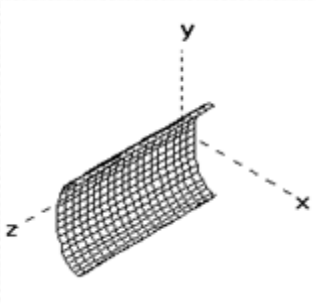
- Se poate folosi orice curba de forma libera pt care exista o ec. matriciala: Hermite, cubica Bezier, segment de curba B-spline, segment de curba spline de interpolare.

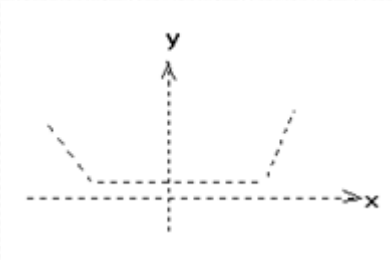
Suprafete de translatie (1)



Arc de elipsa

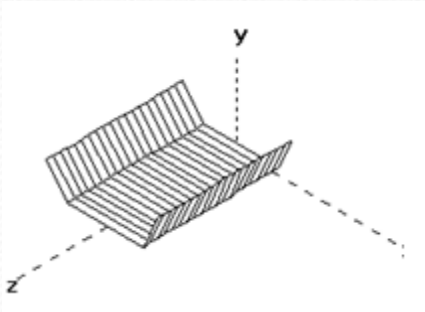
Translatie cu o distanta d
de-a lungul axei OZ





Polilinie in planul XOY

Translatie cu o distanta d
de-a lungul axei OZ



Suprafete de translatie(2)

Translatia unui segment de dreapta

p(u) = P1 + u (P2 - P1) = [(x1+u(x2-x1)) (y1+u(y2-y1)) (z1+u(z2-z1)) 1]

cu u $\in$ [0 , 1]

$$T(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & d \cdot t & 1 \end{bmatrix}$$

Translatia de-a lungul axei OZ cu o distanta d

t $\in$ [0 , 1];

Traiectoria spatiala este un segment de dreapta S1-S2:

s(t) = S1 + t(S2-S1) 0<= t <= 1

Ec. suprafetei de translatie

*S(u, t) = p(u) * T(t)* , cu u $\in$ [0 , 1] si t $\in$ [0 , 1]

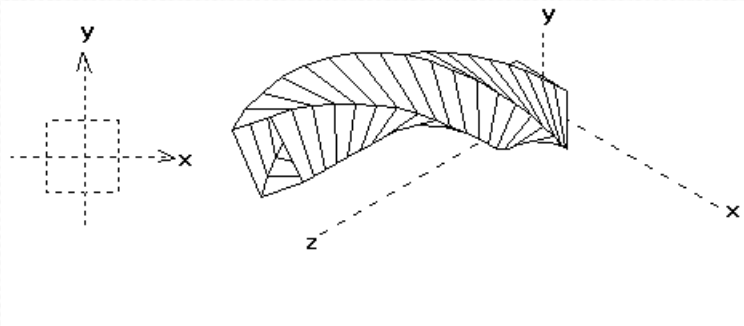
= [x(u) y(u) z(u) + d*t 1]

Suprafete de translatie si rotatie

Translatie de-a lungul axei OZ si rotatie in jurul axei OZ, in fiecare pas:

$$T(t) = \begin{bmatrix} \cos(2\pi t) & \sin(2\pi t) & 0 & 0 \\ -\sin(2\pi t) & \cos(2\pi t) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & d \cdot t & 1 \end{bmatrix}$$

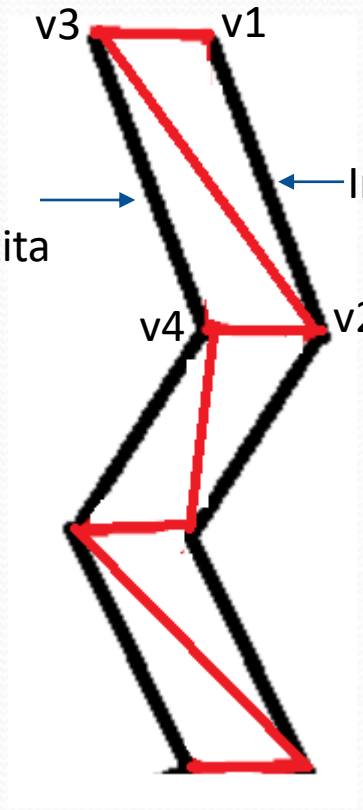
$t \in [0, 1];$



Afisarea unei suprafete de rotatie → laboratorul 2: folosind Geometry shader

Afisarea unei suprafete de baleiere spatiala

Laborator 2



Instanta i+1:
instanta i rotita

Instanta i: o curba Bezier

Intrarea: 4 puncte, P1, P2, P3, P4 ce definesc o curba Bezier – transmise ca uniforme.

Iesirea din Geometry shader: triangle_strip triunghiurile (v1,v2,v3), (v2,v4,v3),…. care unesc puncte ale instantelor i si i+1.

Bibligrafie pentru capitolul “Suprafete de baleiere spatiala”

1. Florica Moldoveanu, Z. Racovia, G. Hera, S. Petrescu, M. Zaharia, *Grafica pe Calculator, Ed. Teora, Bucuresti 1996.*
2. Michael E. Mortenson, *Geometric Modeling, Ed. John Wiley & Sons, 1985.*
3. *D. F. Rogers and J. A. Adams, Mathematical Elements for Computer* Graphics, 2nd. Edition, McGraw-Hill International Edition, 1990.
4. Curves and Surfaces for Computer Graphics, David Solomon, Springer, 2006, ISBN -10: 0-387-24196
5. http://en.wikipedia.org/wiki/Freeform_surface_modelling
6. http://www.scribd.com/doc/57064617/69/Sweep-Surfaces

Suprafete de forma libera -1

Prof. univ. dr. ing. Florica Moldoveanu

Curs *Sisteme de Prelucrare Grafică* – UPB, Automatică şi Calculatoare
2021-2022

Suprafete de forma libera(1)

Definite prin:

- "puncte de control" (puncte din spatiul 3D) si, in unele cazuri, alte conditii geometrice asociate punctelor de control

  Exemple: suprafete Hermite, Bézier, B-spline, Nurbs
- curbe 3D de forma libera

  Exemple: suprafete riglate, suprafete cilindrice

2 tipuri de suprafete de forma libera definite prin puncte de control:

- suprafete de interpolare: contin toate punctele de control
- suprafete de aproximare: nu contin toate punctele de control

Modelarea matematica: foloseste notiunea de "patch" (petic de suprafata)

Suprafete de forma libera(2)

Un petic de suprafata (patch) este o colectie de puncte marginita de 4 curbe.

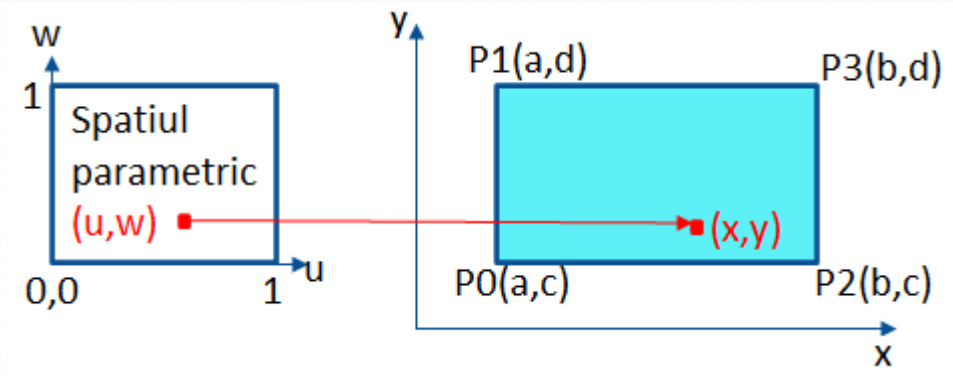
Coordonatele punctelor sunt definite prin functii continue de 2 variabile:

x(u,w), y(u,w), z(u,w)

Ecuatia unui "petic" de suprafață:

p(u,w)= [x(u,w), y(u,w) z(u,w)] cu u1 <= u <=u2 și w1 <= w <= w2

O suprafata dreptunghiulara din planul XOY este un petic de suprafata plana:



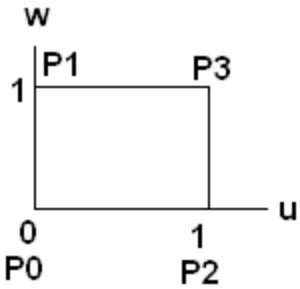
Reprezentarea parametrica
a suprafetei dreptunghiulare

x(u,w) =P0.x + u(P2.x – P0.x)
y(u,w)= P0.y + w(P1.y – P0.y)
z(u,w) = 0
0<= u,w <=1

Orice punct al suprafeței se obtine interpoland biliniar intre laturile dreptunghiului.

Dreptunghiul este un caz particular de suprafata biliniara.

Suprafete biliniare (1)



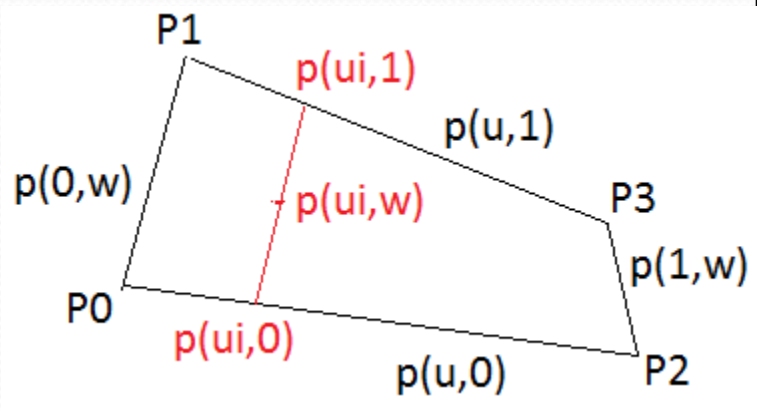
O suprafață biliniară este definită prin patru puncte situate arbitrar in spatiu, care se asociaza colturilor patratului unitate din spatiul parametric (u,w):

P0 = p(0,0); P1 = p(0,1); P2 = p(1,0); P3 = p(1,1)

Frontierele peticului de suprafata biliniara sunt segmentele de dreapta care unesc cele 4 puncte:

p(u,0) = P0 + (P2-P0)u; p(u,1) = P1 + (P3-P1)u, 0<=u<=1

p(0,w), p(1,w), 0<=w<=1



- Punctele suprafetei se obtin prin interpolare biliniara

între punctele de pe frontiere.

- Punctele p(ui,0) si p(ui,1) se obtin interpoland liniar

intre P0 si P2, respectiv intre P1 si P3, pe axa u.

- Punctele dintre p(ui,0) - p(ui,1) se obtin prin interpolare liniara intre cele 2 puncte, pe axa w:

p(ui,0) = P0 + (P2-P0)ui = P0(1-ui) + P2ui; p(ui,1) = P1(1-ui) + P3ui

p(ui,w) = p(ui,0) + (p(ui,1) – p(ui,0))w = P0(1-ui) + P2ui + (P1(1-ui) – P0(1-ui) – P2ui +P3ui)w

0<=w<=1

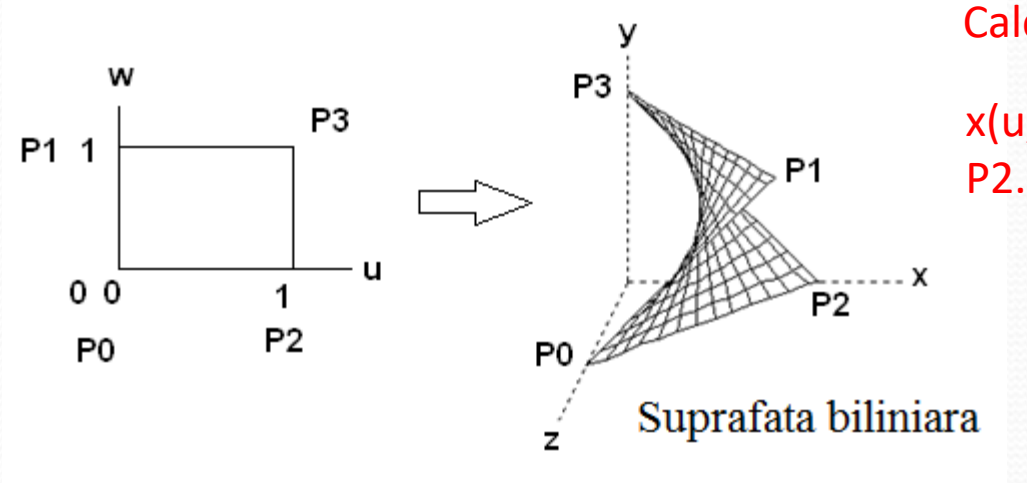
Suprafete biliniare (2)

Se obtine ecuatia suprafetei înlocuindu-l pe ui cu u:

p(u,w) = P0(1-u) + P2u + (P1(1-u) – P0(1-u) – P2u +P3u)w

= P0(1-u)(1-w) + P1(1-u)w + P2u(1-w) + P3uw 0<= u, w <=1

Exemplu: suprafata biliniara data prin punctele P0(0,0,1), P1(1,1,1), P2(1,0,0), P3(0,1,0)



Calculul punctelor de pe suprafata:

x(u,w) = P0.x*(1-u)(1-w) + P1.x*(1-u)w + P2.x*u(1-w) + P3.x*uw

x(u,w) = (1-u)*w + u*(1-w)
y(u,w) = (1-u)*w + u*w
z(u,w) =(1-u)*(1-w) + (1-u)*w

Frontierele suprafetei biliniare sunt segmente de dreapta → peticele biliniare nu sunt adecvate pentru modelarea suprafetelor 3D curbe, deoarece nu se poate asigura continuitatea geometrica la jonctiunea a 2 petice biliniare.

Suprafete biliniare (3)

- Daca 2 dintre cele 4 puncte care definesc suprafata coincid se obtine un petic de suprafata triunghiular. Exemplu: P0 =(0, 0, 1), P2 = (1, 0, 0), P1 = P3 = (0, 1, 0)

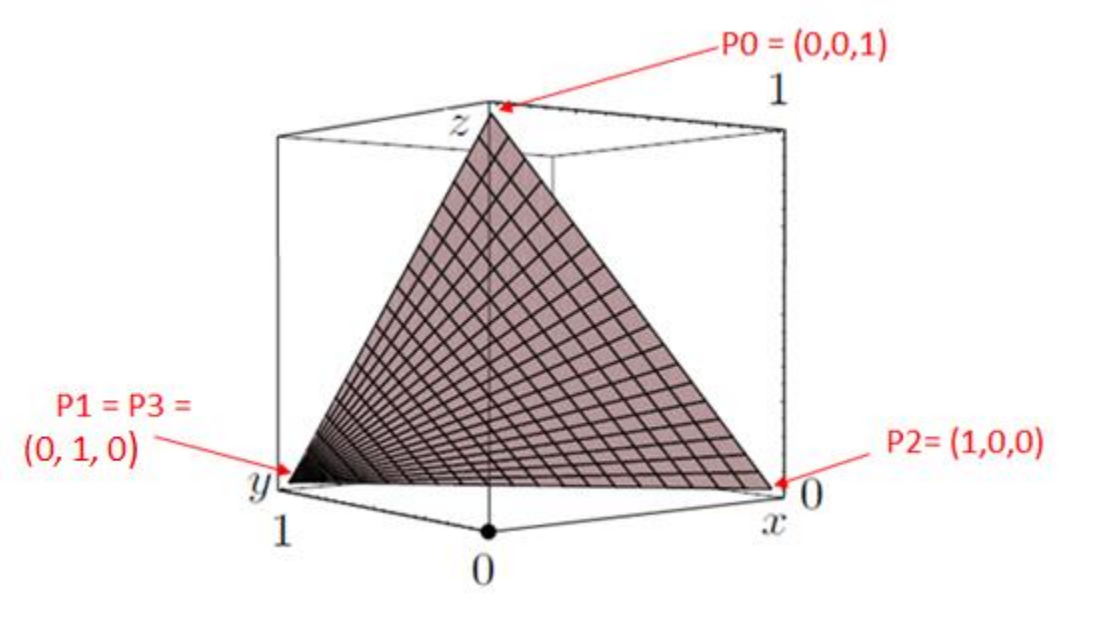
- Ecuatia suprafetei este:

$p(u,w) = [0, 0, 1](1-u)(1-w) + [0, 1, 0](1-u)w + [1, 0, 0]u(1-w) + [0, 1, 0]uw \qquad 0<=u,w<=1$

x(u,w) = P0.x *(1−*u*)*(1−*w*) + P1.x*(1-u)*w + P2.x*u*(1-w) + P3.x*u*w = u*(1-w)

y(u,w) = P0.y *(1−*u*)*(1−*w*) + P1.y*(1-u)*w + P2.y*u*(1-w) + P3.y*u*w = (1-u)*w +u*w

z(u,w) = P0.z *(1−*u*)*(1−*w*) + P1.z*(1-u)w + P2.z*u*(1-w) + P3.z*u*w = (1-u)*(1-w)



Vizualizarea suprafetelor de forma libera

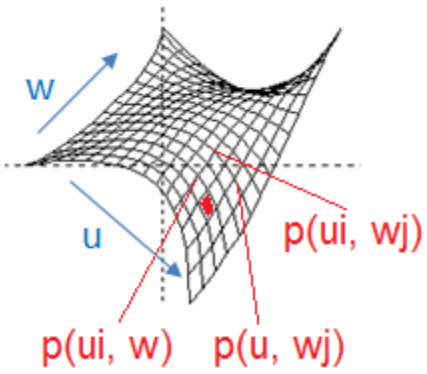
Prof. univ. dr. ing. Florica Moldoveanu

Curs *Sisteme de Prelucrare Grafică* – UPB, Automatică şi Calculatoare
2021-2022

Vizualizarea suprafetelor de forma libera

- **Afişare wireframe**
  - Suprafata este aproximata prin doua familii de curbe ortogonale in spatiul parametric (curbe de u=const si curbe de w=const), care unesc varfurile unei grile regulate de puncte de pe suprafaţă:

$p(u_i, w_j)$, $0 <= u_i, w_j <= 1$, $u_{i+1} = u_i + du$, $w_{j+1} = w_j + dw$



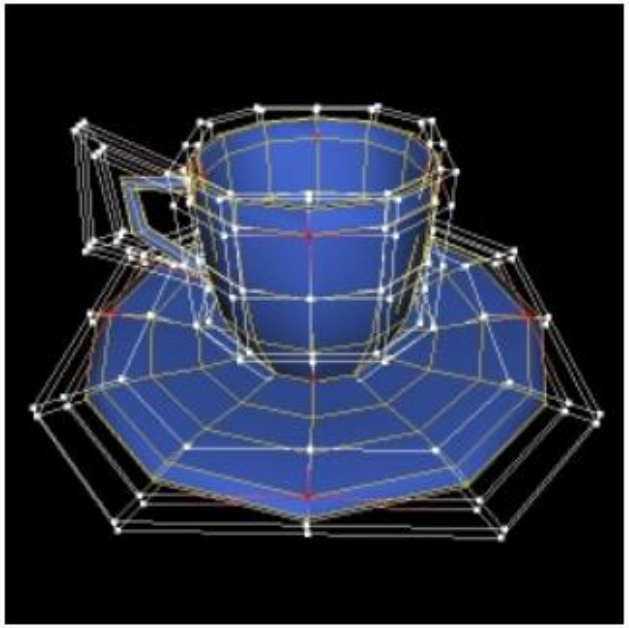
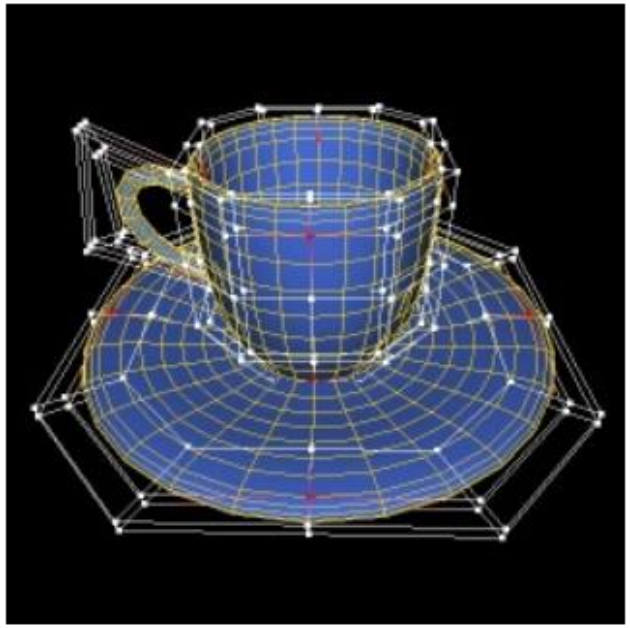
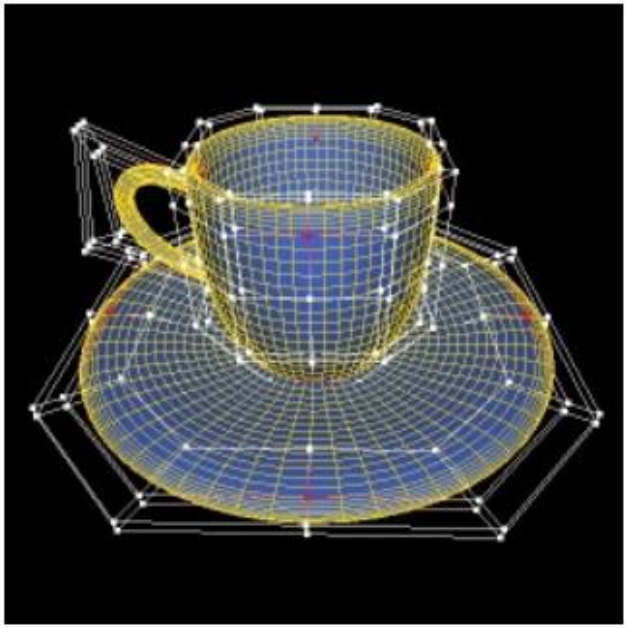
- **Afişare prin feţe opace**

1. Suprafata este aproximata printr-o rețea de fețe care conecteaza varfurile unei grile regulate de puncte de pe suprafaţă: $p(u_i, w_j)$, $0 <= u_i, w_j <= 1$, $u_{i+1} = u_i + du$, $w_{j+1} = w_j + dw$

   Fiecare faţa conecteaza 4 varfuri vecine ale grilei, este considerata plată şi aproximată la afişare printr-un patrulater sau 2 triunghiuri.

2. Suprafata este aproximata printr-o reţea de feţe plane obţinută prin divizarea suprafetei.

Vizualizarea suprafetelor de forma libera

http://www.visualizationlibrary.org/documentation/pag_guide_bezier_surfaces.html

Ceasca de cafea modelata cu 26 petice bicubice Bezier, vizualizata la trei niveluri de netezire. De la stânga la dreapta creşte numărul de fețe prin care este aproximată suprafața.

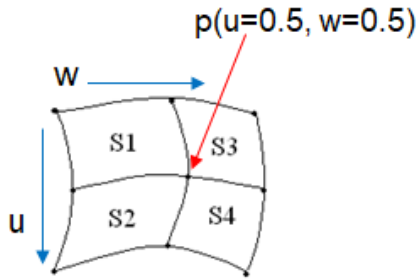
Alb: punctele de control si poliedrul de control

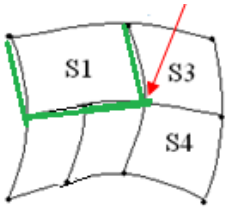
Rosu: colturile peticelor

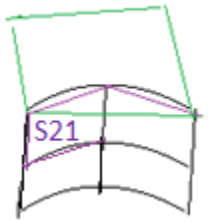
Afisarea unei suprafete Bezier printr-o retea de fatete plane

- Suprafata este descompusa în segmente de suprafață care satisfac un criteriu de planeitate

- Divizarea este recursiva, din fiecare divizare rezultand 4 segmente de suprafață







- Se verifica planeitatea fiecarui segment rezultat din divizarea curenta: daca un segment nu este plan, se continua divizarea.

- Segmentele de suprafata plane sunt aproximate prin patrulatere sau triunghiuri plane, care sunt trimise in banda grafica.

➢Divizarea este neuniforma: divizarea unui segment de suprafata se opreste daca satisface criteriul de planeitate impus; se mai numeste divizare adaptiva.

- mai eficienta decat divizarea uniforma a spatiului parametric: pot rezulta mai putine fatete
- pot sa apara defecte la frontiera a doua fațete care corespund la nivele diferite de divizare

Divizarea adaptiva a unui petic bicubic Bézier

```
void DivideSuprafata(Mat S)
{  // S este matricea punctelor de control (4x4)
   if (suprafata definita prin S este plana)
   { * trimite în banda grafică patrulaterul (descompus in triunghiuri) determinat de colțurile
             suprafetei S;
      return;
   }
  * se divide suprafața definita prin S in punctul p(u=0.5, w=0.5) rezultand suprafețele
      definite prin matricile punctelor de control S1, S2, S3, S4;
   DivideSuprafata(S1); DivideSuprafata(S2); DivideSuprafata(S3); DivideSuprafata(S4);
}
```

Criterii de planeitate pentru un petic de suprafata Bezier

1) Coplanaritatea varfurilor poliedrului caracteristic al peticului, cu o anumita toleranță:

- peticul Bezier este inclus in poliedrul convex al punctelor de control (închidere convexă)

- Se calculeaza planul in care se afla 3 dintre colturile peticului.
- Se calculeaza distanta maxima de la celelalte puncte de control la acest plan.
- Daca această distanță este sub o limita de toleranță, peticul este considerat plan.

2) **Coplanaritatea colturilor peticului**

si liniaritatea marginilor peticului.

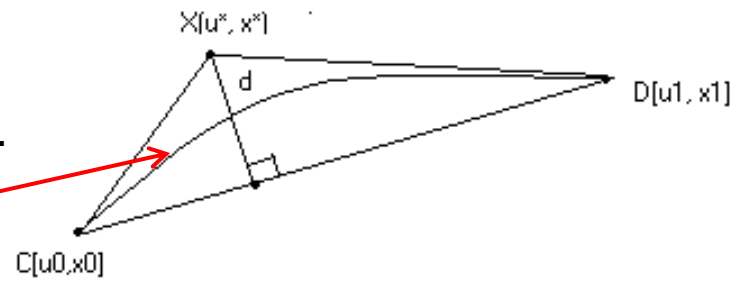
$$D = \begin{vmatrix} x_0 & y_0 & z_0 & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{vmatrix} < \delta$$

- Frontierele peticului bicubic Bezier sunt curbe cubice,

p(u)= (x(u), y(u), z(u))

- Verificarea liniaritatii se face separat pe axele x, y, z.

Exemplu: x(u)



- Daca d<toleranţa, frontiera este liniara.

Afisarea unei bicubice Bezier/Bspline folosind o grila regulata de puncte de pe suprafata

Ecuatia unei suprafete bicubice Bézier

$p(u, w) = U^* M_B^* B^* M_B^T{}^* W^T = U^* M^* W^T$, $0 <= u <= 1$, $0 <= w <= 1$

B – matricea punctelor de control ale suprafetei (4x4)

M_B – matricea de baza Bézier

Ec. unui segment de suprafata bicubica B-spline

$p_{st}(u, w) = U^* M_{s4}^* P_{4,4}(s,t)^* M_{s4}^T{}^* W^T = U^* M^* W^T$, $0 <= u <= 1$, $0 <= w <= 1$

$P_{4,4}$ este matricea punctelor de control care determina segmentul de suprafață

Discretizarea spatiului parametric:

vec3 PS[NU, NW]; // NU, NW – constante, matricea punctelor de pe suprafata

du= 1.0/(NU-1); dw = 1.0/(NW -1); //paşii de discretizare

Afisarea unei bicubice Bezier folosind o grila regulata de puncte de pe suprafata

```
void afisareSuprafBezier()
{   vec3 M[4][4] = CalculM(M_B,B); // calculeaza vec3 M[4][4] = M_B*B*M_B^T
    CalculPunctePeticBicubic(M); // calculeaza puncte p(u,w) si le memoreaza in matricea PS
   * afişare suprafata Bezier prin 2 familii de curbe sau fațete opace folosind matricea PS
}
void CalculPunctePeticBicubic(vec3 M[4][4])
{ // p( u, w) = U*M* W^T   U = [u^3  u^2  u  1], W = [w^3  w^2  w  1]
   for( u=0, iu=0; iu<=NU-1; u+ = du, iu++)
     for(w=0, jw=0; jw<=NW-1; w+ = dw, jw++)
            PS[iu][jw] = w*w*w*(u*u*u* M[0][0] + u*u*M[1][0] + u*M[2][0] + M[3][0]) +
                         w*w* (u*u*u* M[0][1] + u*u*M[1][1] + u*M[2][1] + M[3][1]) +
                         w*(u*u*u* M[0][2] + u*u*M[1][2] + u*M[2][2] + M[3][2]) +
                         u*u*u* M[0][3] + u*u*M[1][3] + u*M[2][3] + M[3][3];
 }
```

Afisarea unei bicubice B-spline folosind o grila regulata de puncte de pe suprafata

void afisareB-spline()

{ // afiseaza o suprafata B-spline bicubica definita prin (n+1)x(m+1) puncte de control

// pentru fiecare segment de suprafata

// p_{st}(u, w) = U* M_{s4}*$P_{4,4}$(s,t)*M_{s4}^T *W^T = U*M* W^T

for(int s=1; s<= m–2 ; s++)

for(int t=1; t <= n–2; t++)

{ * calculeaza matricea $P_{4,4}$(s,t) // 4x 4 puncte de control

vec3 M[4][4] = CalculM(M_{s4}, $P_{4,4}$); // calculeaza vec3 M[4][4] = M_{s4}*$P_{4,4}$(s,t)*M_{s4}^T

CalculPunctePeticBicubic(M); // aceeasi functie folosita si la Bezier

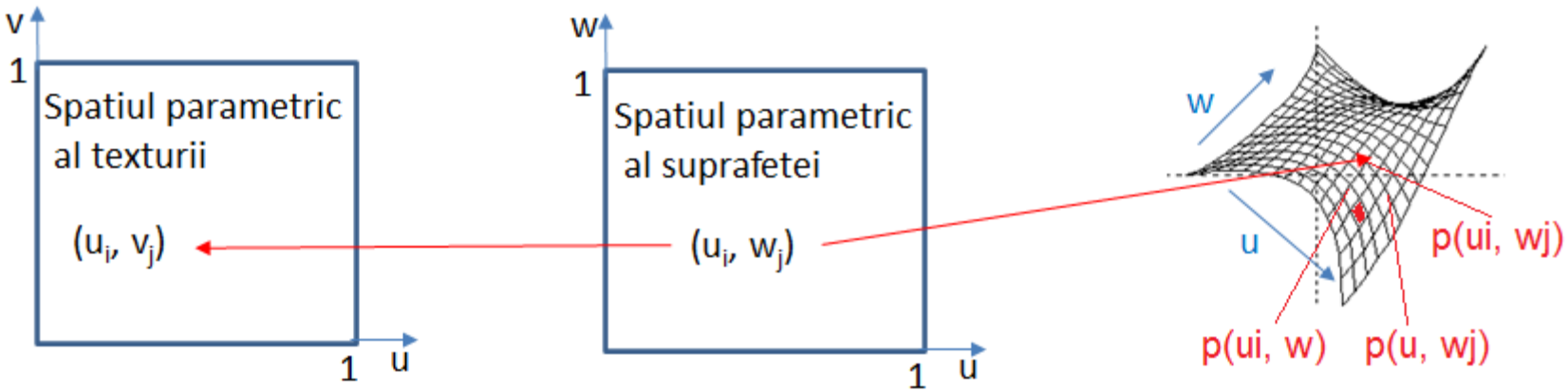
* afisare segment de suprafata folosind matricea de puncte PS

}

}

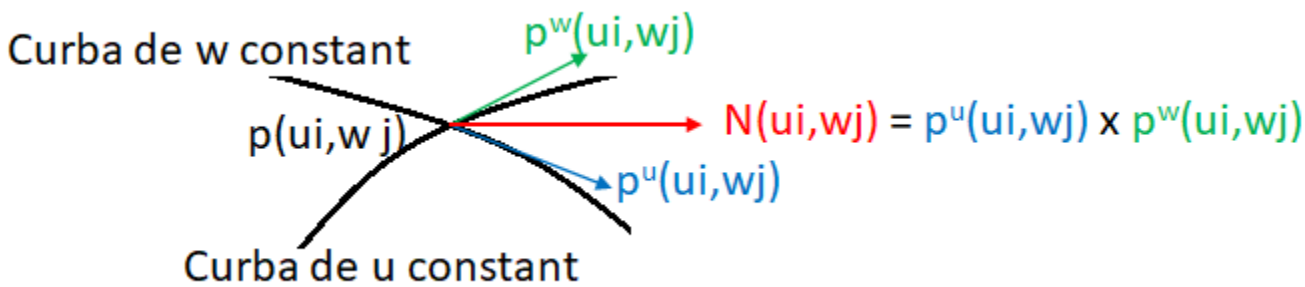
Afisarea unui petic bicubic Bezier cu calcul de iluminare si textura

- Pentru calculul reflexiei luminii in punctele suprafetei este necesar sa se ataseze fiecarui varf calculat, $p(u_i, w_j)$, normala in acel punct.
- Daca pe suprafata peticului trebuie aplicata o textura, se asociaza fiecarui varf al mesh-ului valorile (u_i, w_j) corespunzatoare varfului, acestea fiind folosite pentru adresarea spatiului parametric al texturii.



Calculul normalei într-un punct al grilei(1)

- Normala într-un punct p(ui,wj) se calculează ca produs vectorial al vectorilor tangenți la suprafață în p(ui,wj), pe direcția u, respectiv w.



$p(u,w) = \sum\sum P_{i,j} * B_{i,3}(u) * B_{j,3}(w)$ $\quad 0<=i, j<=3 \quad 0<= u, w <= 1$

$$\frac{\partial}{\partial u} p(u,w) = B'_{0,3}(u) \sum_j B_{j,3}(w) * P_{0j} + B'_{1,3}(u) \sum_j B_{j,3}(w) * P_{1j} + B'_{2,3}(u) \sum_j B_{j,3}(w) * P_{2j} + B'_{3,3}(u) \sum_j B_{j,3}(w) * P_{3j}$$

$B_{0,3}(u) = (1-u)^3$

$B_{1,3}(u) = 3*u*(1-u)^2$

$B_{2,3}(u) = 3*u*(1-u)$

$B_{3,3}(u) = u^3$

$B'_{0,3}(u) = -3(1-u)^2$

$B'_{1,3}(u) = 3(1-u)^2 - 6u(1-u)$

$B'_{2,3}(u) = 6u(1-u) - 3u^2$

$B'_{3,3}(u) = 3u^2$

Calculul normalei într-un punct al grilei(2)

Analog se obține $\frac{\partial}{\partial w} p(u,w)$, în care apar $B'_{j,3}(w)$

Fie

vec3 bezier(vec3 P0, vec3 P1, vec3 P2, vec3 P3, float t);

functia care intoarce punctul de pe curba cubica Bezier data prin P0, P1, P2, P3, pentru valoarea lui t

vec3 duBez(vec3 control[4][4], float u, float w);

funcția care calculeaza vectorul tangent in punctul (u,w) pe directia u

vec3 dwBez(vec3 control[4][4], float u, float w);

funcția care calculeaza vectorul tangent in punctul (u,w) pe directia w

Calculul normalei într-un punct al grilei(3)

$$\frac{\partial}{\partial u} p(u,w) = B'_{0,3}(u) \sum_j B_{j,3}(w) * P_{0j} + B'_{1,3}(u) \sum_j B_{j,3}(w) * P_{1j} + B'_{2,3}(u) \sum_j B_{j,3}(w) * P_{2j} + B'_{3,3}(u) \sum_j B_{j,3}(w) * P_{3j}$$

```
vec3 duBez(vec3 control[4][4], float u, float w)
{//control: matricea punctelor de control pentru peticul Bezier
return
 -3*(1-u)*(1-u)* bezier(control[0][0], control[0][1], control[0][2], control[0][3], w)+
 (3*(1-u)*(1-u)-6*u*(1-u))* bezier(control[1][0], control[1][1], control[1][2], control[1][3], w)+
 (6*u*(1-u)-3*u*u)*bezier(control[2][0], control[2][1], control[2][2], control[2][3], w)+
  3*u*u* bezier(control[3][0], control[3][1], control[3][2], control[3][3], w);
}
```

Analog se defineste

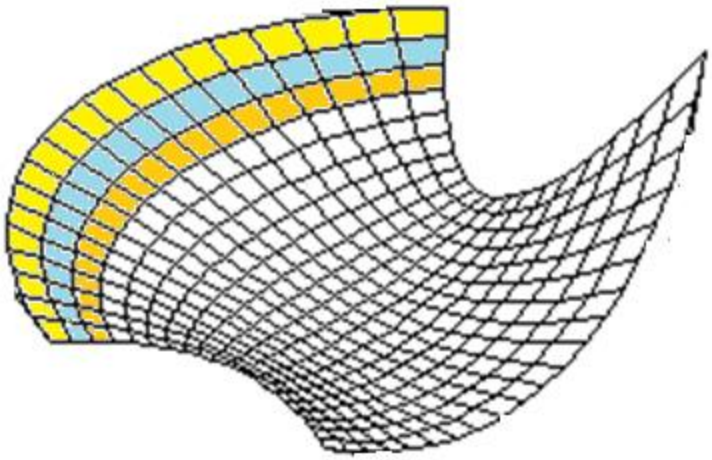
```
vec3 dwBez(vec3 mat[4][4], float u, float w);
```

Calculul grilei de puncte cu normale si coordonate textura

```
vec3  PS[NU, NW]; // NU, NW – constante, matricea punctelor de pe suprafata
vec3  NS[NU, NW]; // matricea normalelor in punctele calculate de pe suprafata
vec2  tex[NU][NW]; //matricea coordonatelor textura ale punctelor calculate de pe suprafeta

// Calculează vârfuri mesh prin care se afişează suprafața,
// asociaza normala şi coordonatele textură fiecărui vârf
void CalculBezierIlumTex()
{vec3 M[4][4] = CalculM(M_B,B); // calculeaza vec3 M[4][4] = M_B*B*M_B^T
 for( u=0, iu=0; u<=1; u+ = du, iu++)
    for(w=0, jw=0; w<=1; w+ = dw, jw++)
      {   PS[iu][jw] = …….//la fel ca în CalculPunctePeticBicubic()
          vec3 du = duBez(B, u, w); //tangenta pe directia u in punctul p(u,w)
          vec3 dw = dwBez(B, u, w); //tangenta pe directia w in punctul p(u,w)
          NS[iu][jw] = normalize(cross(du, dw));//intoarce produsul vectorial al vectorilor du, dw
          tex[iu][jw].x = u; tex[iu][jw].y = w;
      }
}
```

Afisarea unui petic bicubic Bezier folosind un Geometry Shader

- La un apel al programului GS se creaza o fîşie a suprafetei, cuprinsa intre 2 curbe de u constant
- Iesirea din GS este o primitiva triangle_strip.
- Pentru aproximarea suprafetei prin n fîşii se trimit in banda grafica n instante ale unei primitive linie, care nu este folosita in GS (la fel ca in programul de generare a unei suprafete de rotatie)
- Pentru generarea celor n instante se foloseste **glDrawElementsInstanced.**



Varfuri generate la un apel la programului Geometry Shader:

for (int index = 0; index < no_of_generated_points; index ++)
index = 0: v0, v1
index = 1: v2, v3
……
index = no_of_generated_points – 1: vn, vm

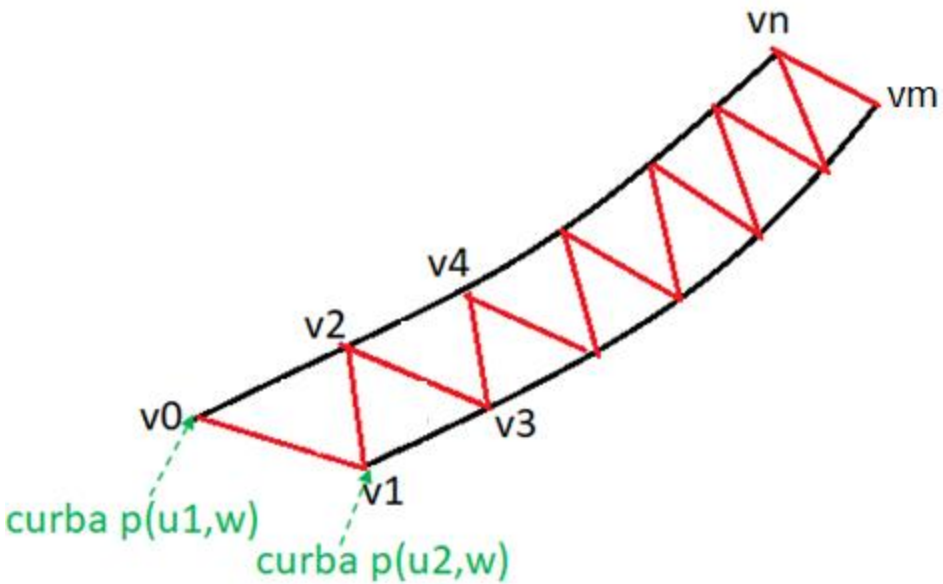
v0, v2, v4, …sunt puncte succesive de pe curba p(u1,w) calculate cu pasul dw

v1, v3, v5, …sunt puncte succesive de pe curba p(u2,w) calculate cu pasul dw

u1 = nr_instanta_crt * du
u2 = (nr_instanta_crt+1)* du

Iesirea din Geometry shader:
o primitiva triangle_strip.

Toate fatetele triunghiulare ale primitivei sunt orientate trigonometric:
(v0, v1,v2), (v2,v1,v3),…



Afisarea unui petic bicubic Bezier folosind un Geometry Shader

Bezier.cpp

Se apeleaza

glDrawElementsInstanced(GL_LINES, static_cast<int>(mesh->indices.size()),

GL_UNSIGNED_INT, (void*)0, instances)

- Pentru a se desena primitiva “mesh” de un numar de ori: instances
- Primitiva este ignorata in Geometry shader
- In GS se deseneaza o fîşie a suprafetei Bezier

Vertex shader

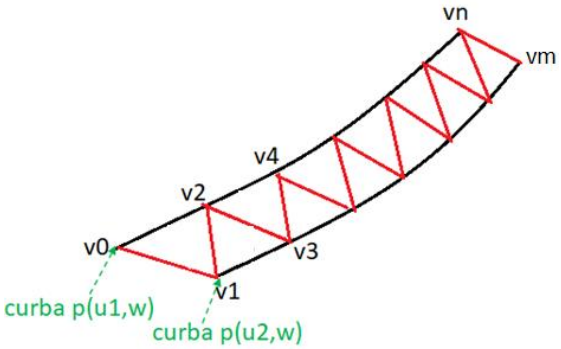
```
// apelat pentru fiecare varf al primitivei instantiate (o linie)
layout(location = 0) in vec3 v_position;

uniform mat4 Model;

out int instance; //numarul instantei din care face parte varful curent

void main()
{
instance = gl_InstanceID; //numarul instantei din care face parte varful
gl_Position =  Model * vec4(v_position, 1);
}
```

Geometry shader (1)

```
layout(lines) in;//primitiva care ar trebui desenata
layout(triangle_strip, max_vertices = 256) out;

uniform mat4 View;
uniform mat4 Projection;
uniform int no_of_instances; //numarul de fasii prin care se aprox suprafata
uniform int no_of_generated_points; //puncte generate de pe o curba Bezier p(u,w)
uniform vec3 M[4][4]; // M = M_B*B*M_B^T
uniform vec3 controlPoints[4][4]; //punctele de control pt suprafata Bezier

//intrari pentru fragment shader
out vec3 world_position;
out vec3 world_normal;
out vec2 texcoord;

in int instance[2]; //ambele contin aceeasi valoare: nr. instantei curente, setata in Vertex shader

vec3 Bezier1(vec3 M[4][4], float u, float w);//intoarce un punct de pe suprafata Bezier

vec3 bezier(vec3 p1, vec3 p2, vec3 p3, vec3 p4, float t); //intoarce un punct de pe curba Bezier

vec3 duBez(vec3 controlPoints[4][4], float u, float w); //intoarce valoarea derivatei in raport cu u

vec3 dwBez(vec3 controlPoints[4][4], float u, float w); //intoarce valoarea derivatei in raport cu w
```

Geometry shader (2)

```
void main()
{  float du = 1.0f / no_of_instances; float dw = 1.0f / (no_of_generated_points - 1);
   float u1 = instance [0] * du; float u2 = (instance [0] + 1) * du;

   for (int index = 0; index < no_of_generated_points; index ++)
   {    float w = index * dw;
        world_position = Bezier1 (M, u1, w);
        vec3 tangent1 = duBez (controlPoints, u1, w);
        vec3 tangent2 = dwBez (controlPoints, u1, w);
        world_normal = normalize (cross (tangent1, tangent2));
        texcoord = vec2 (u1, w);
        gl_Position = Projection* View * vec4(world_position, 1);
        EmitVertex();
       world_position = Bezier1 (M, u2, w);
       tangent1 = duBez (controlPoints, u2, w);
       tangent2 = dwBez (controlPoints, u2, w);
       world_normal = cross (tangent1, tangent2);
       texcoord = vec2 (u2, w);
       gl_Position = Projection* View * vec4(world_position, 1);
       EmitVertex();
  } EndPrimitive();
}
```



Fragment shader

```
#version 430
layout(location = 0) out vec4 out_color;

uniform vec3 light_position;
uniform vec3 eye_position;
uniform sampler2D groundTexture;

in vec3 world_position;
in vec3 world_normal;
in vec2 texcoord;

void main()
{
vec3 normal = normalize (world_normal);
vec3 lightDirection = normalize (light_position - world_position);
vec3 viewDirection = normalize (eye_position - world_position);

……calculeaza culoare fragment……
out_color = ……
}
```

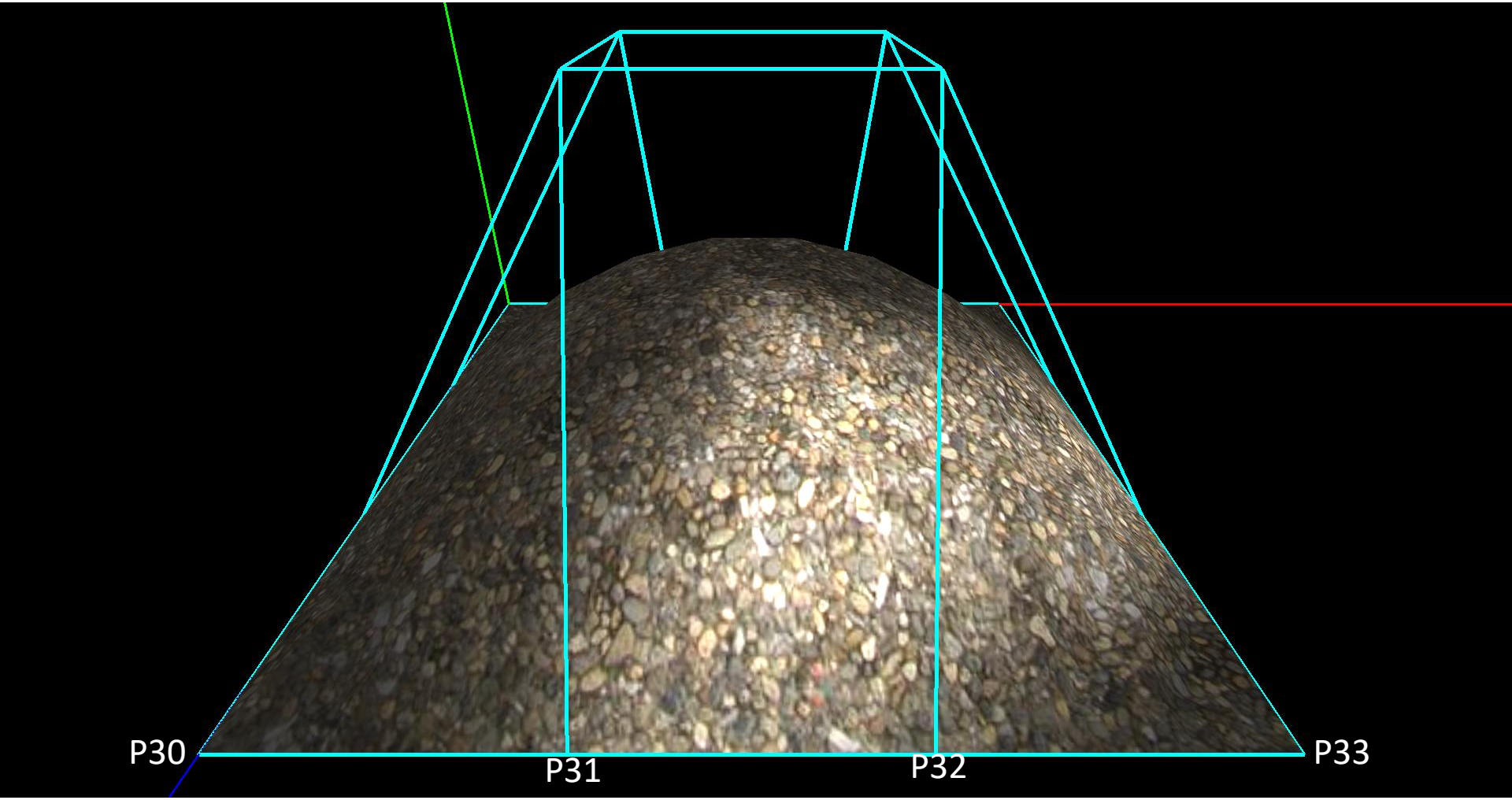
[0 0 0] [3 0 0] [6 0 0] [9 0 0]

[0 0 3] [3 5 3] [6 5 3] [9 0 3]

[0 0 6] [3 5 6] [6 5 6] [9 0 6]

[0 0 9] [3 0 9] [6 0 9] [9 0 9]

Matricea punctelor de control



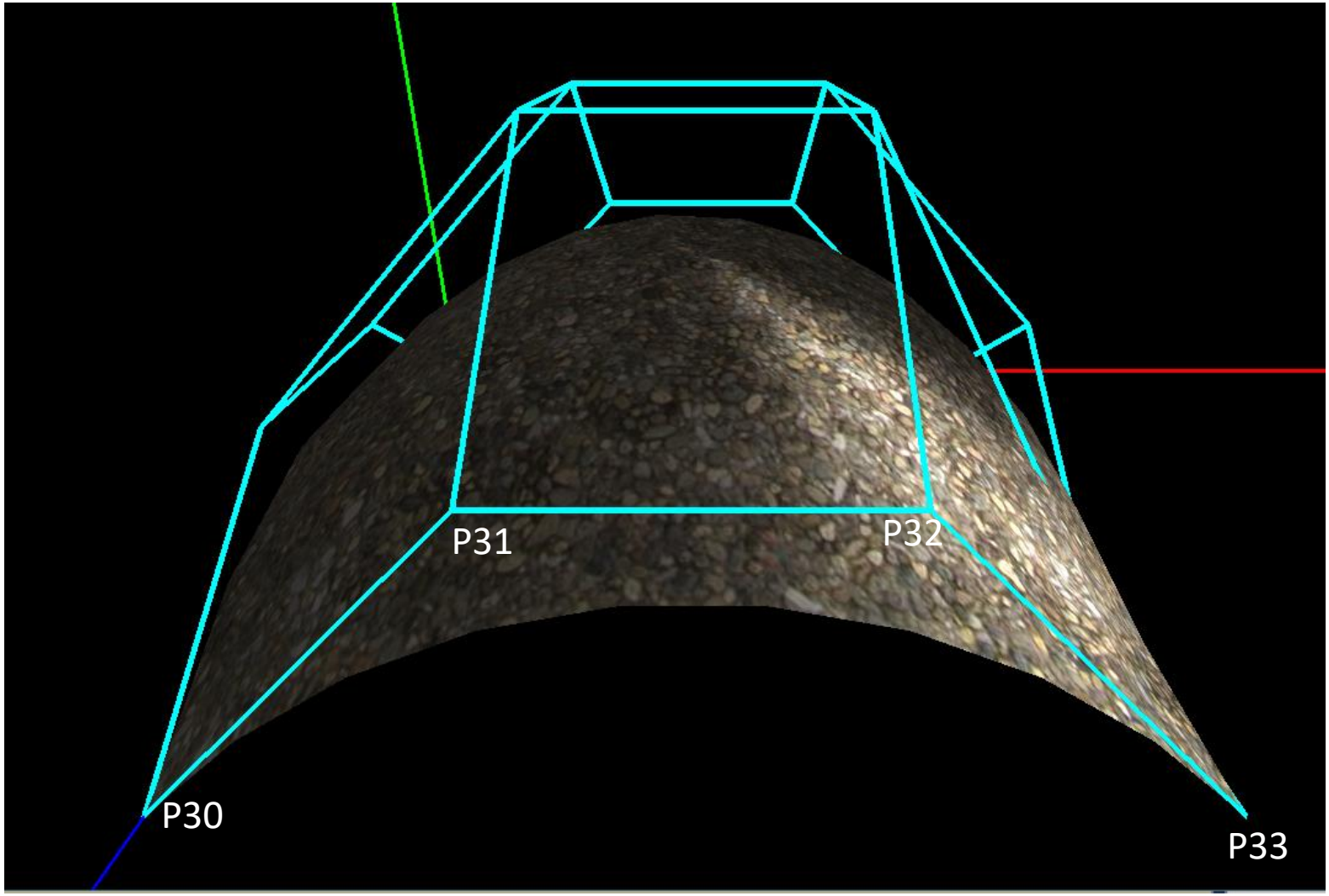
[0 0 0] [3 3 0] [6 3 0] [9 0 0]

[0 2 3] [3 5 3] [6 5 3] [9 2 3]

[0 2 6] [3 5 6] [6 5 6] [9 2 6]

[0 0 9] [3 3 9] [6 3 9] [9 0 9]

Matricea punctelor de control



Redarea suprafetelor 3D folosind texturi - 1

Prof. univ. dr. ing. Florica Moldoveanu

Texturi (1)

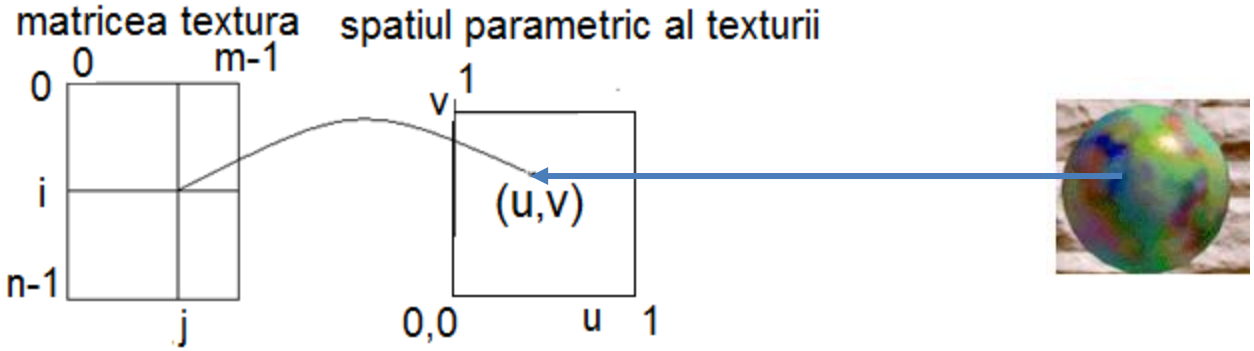
- Se folosesc in diferite scopuri:
  - Redarea unei imagini pe o suprafata

  - Simularea reflexiei/refractiei mediului inconjurator de catre o suprafata
  - Simularea detaliilor geometrice ale unei suprafete
  - Simularea indepartarii fata de observator a obiectelor dintr-o scena 3D, ş.a.

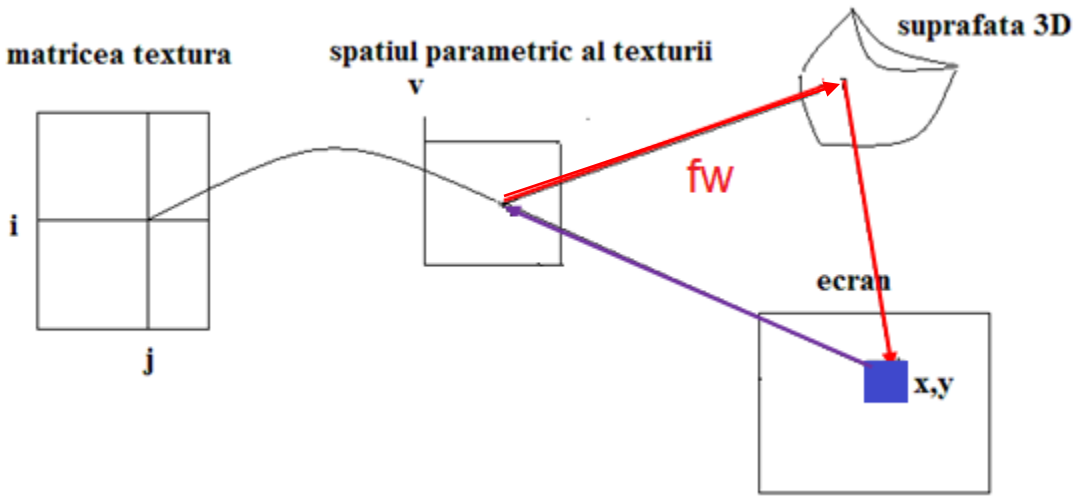
Aplicarea unei imagini textura pe o suprafata 3D

- O imagine este alcatuita din pixeli
- Pixelii imaginii trebuie pusi in corespondenta punctelor suprafetei 3D

- Elementele imaginii textura sunt culori si se numesc « texeli ».
- Corespondenta dintre punctele suprafetei 3D si spatiul discret al texturii este realizata prin intermediul unui spatiu continuu, (u,v), numit *spatiul parametric al texturii,* 0 <= u,v =< 1



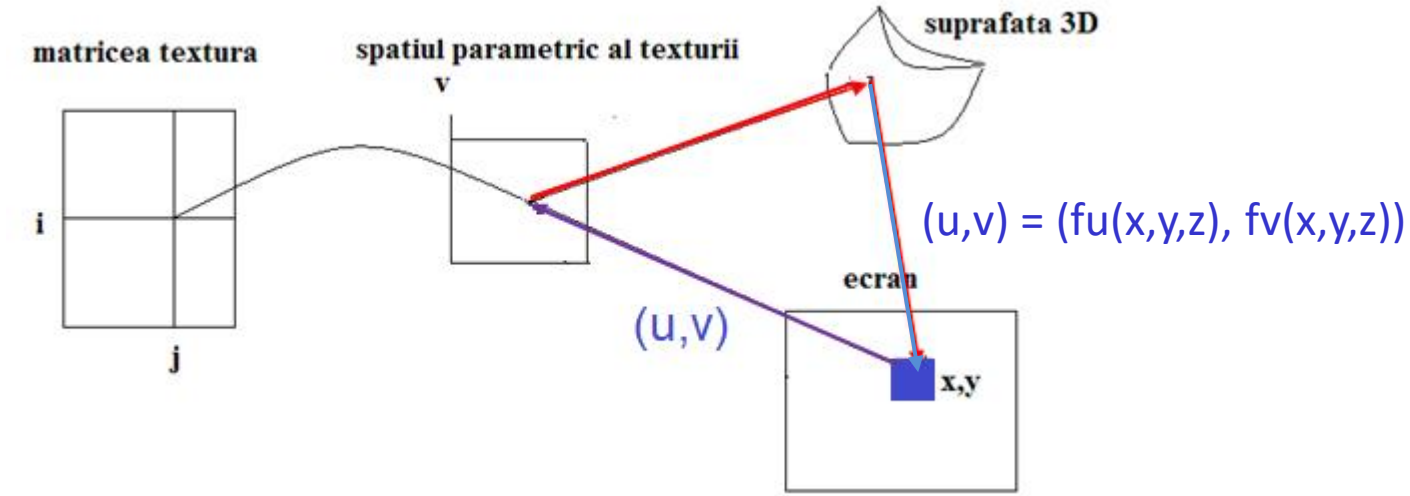
Aplicarea unei imagini textura pe o suprafata 3D - maparea directa (forward mapping)-



Se parcurge imaginea textura linie cu linie si se calculeaza pentru fiecare texel pixelul in care se afiseaza:

- adresa texel, (i,j) → adresa (u,v) → punct de pe suprafaţa 3D (x,y,z) → adresa pixel, (xp, yp)
- Se afiseaza pixelul (xp, yp) in culoarea texelului (i, j)
- Metoda nu este adecvata hardware-ului grafic actual (in plus, poate produce defecte)

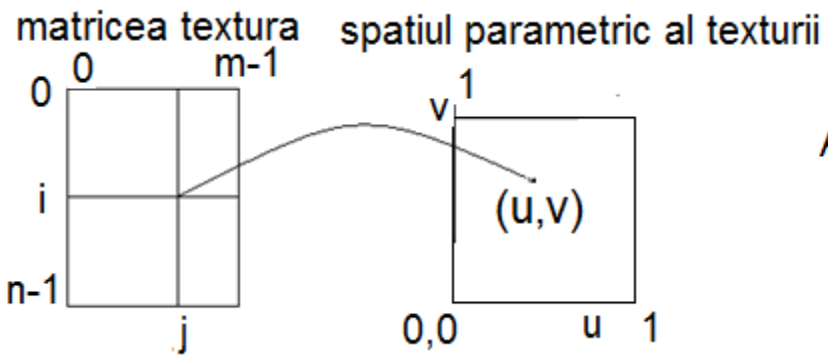
Aplicarea unei imagini textura pe o suprafata 3D - maparea inversa (inverse mapping)-



Se calculeaza pentru fiecare fragment rezultat din rasterizarea suprafetei 3D texelul/texelii din care se obtine culoarea textura.

- Fiecare fragment (xp,yp) are asociata adresa (u,v) corespunzatoare punctului de pe suprafaţa 3D (x,y,z) care se afisaza în (xp,yp); adresa (u,v) → adresa (ir, jr) in spatiul texturii, unde ir, jr sunt numere reale.
- Afiseaza fragmentul folosind culoarea obtinuta de la adresa (ir, jr) prin filtrarea texturii.

Obtinerea unei culori din imaginea textura in maparea inversa



Accesarea matricei textura din spatiul (u,v)

$jr = u * (m-1)$

$ir = (1 - v) * (n-1)$ ⇐ (u, v)

(ir, jr) – numere reale

- Accesarea (matricei) imaginii textura – prin numere intregi

⇓ Filtrarea texturii

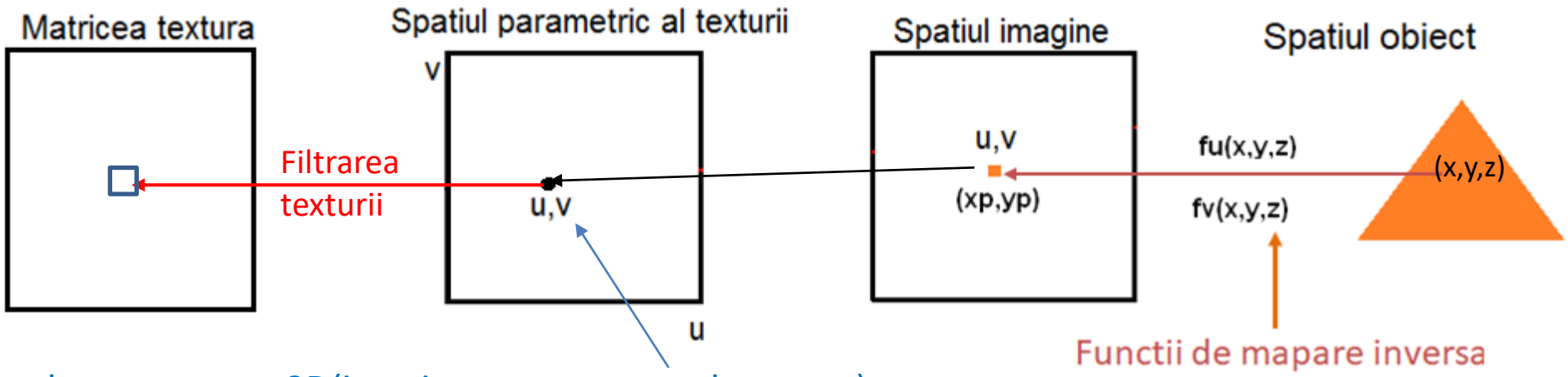
2 posibilitati:

1) Culoare = textura (i,j), unde (i,j) adresa cea mai apropiata de (ir,jr) in spatiul discret al texturii; în OpenGL: GL_NEAREST
2) Culoare = interpolare_biliniara (textura, ir, jr); în OpenGL: GL_LINEAR

Maparea inversa

- Este metoda uzuala de mapare a texturilor pe suprafetele 3D
- Se pleaca de la spatiul imagine, aplicarea texturii fiind inglobata în procesul de rasterizare a primitivelor în care este descompusa suprafata 3D.
- Imaginea suprafatei 3D pe care se aplica textura este generata pixel cu pixel si pentru fiecare pixel (xp, yp) se calculeaza adresa texelului/texelilor care se folosesc in calculul culorii pixelului.

Adresa texelului se obtine folosind **functii de mapare inversa** si « filtrarea texturii ».



culoare = texture2D(imagine_textura, coord_textura)

Functii de mapare inversa

Definesc o corespondenta intre punctele suprafetei 3D si punctele spatiului (u,v) :

$$u = f_u(x,y,z) \; ; \; v = f_v(x,y,z)$$

In general, aceasta corespondenta este calculata pentru varfurile primitivelor rasterizate, adresele (uf,vf) pentru fragmentele rezultate din rasterizare obtinandu-se prin interpolare intre adresele (u,v) asociate varfurilor.

$f_u(x,y,z)$, $f_v(x,y,z)$ Se numesc **functii de mapare inversa**

- Exista functii de mapare inversa numai pentru suprafeţe definite intr-un spatiu parametric, (s,t) a.î. se poate stabili o corespondenta intre spatiul parametric al suprafetei si spatiul parametric al texturii: **(x,y,z) → (s,t) → (u,v)**
- Ele se numesc **funcţii de mapare standard.**

Functii de mapare standard(1)

1. Maparea plana (maparea pe un plan)

Ecuatiile care descriu aceasta mapare sunt:

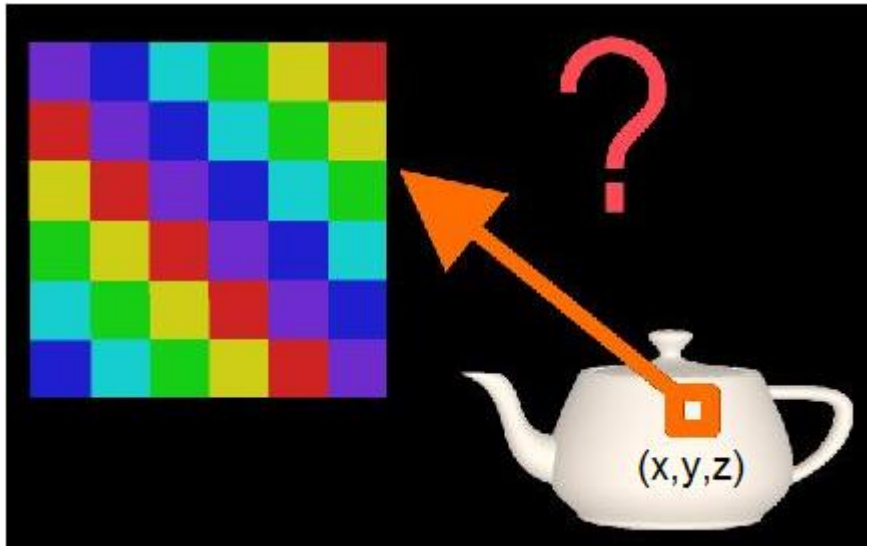
$u = su*x - ou$ $\quad x \in X,\ y \in Y$

$v = sv*y - ov$ $\quad 0 <= u, v <= 1$

unde : su, sv sunt factorii de scalare pentru transformarea spatiului dreptunghiular [X, Y] in [0,1]

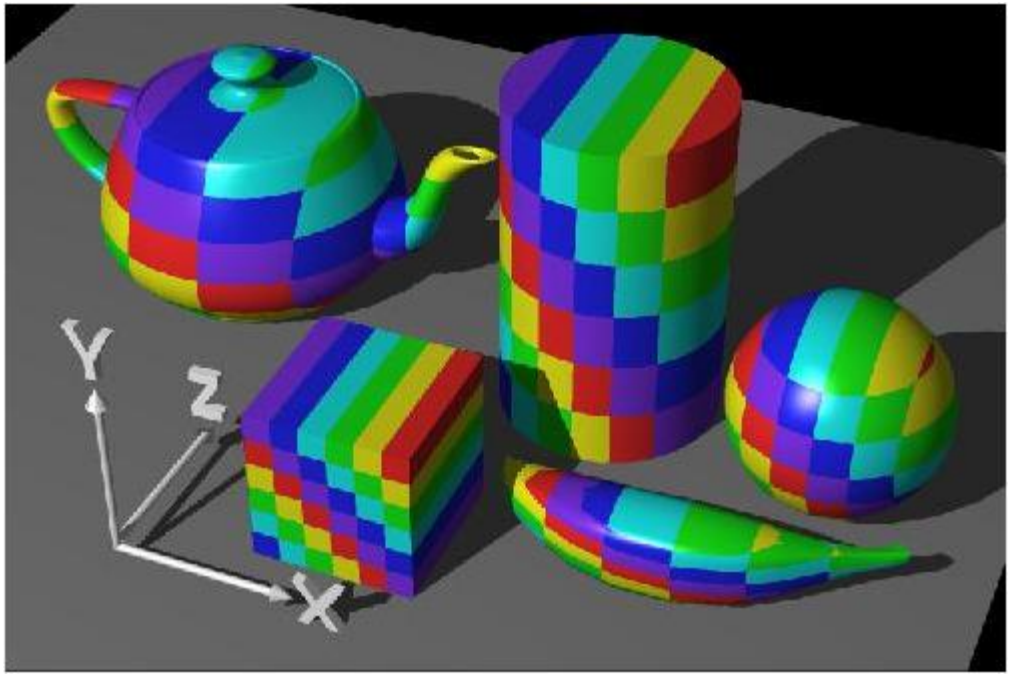
iar (ou,ov) este originea texturii: determina modul în care se aşază textura pe suprafaţă.

(u, v) ← (x, y) ← (x, y, z) - se renunta la coordonata z: suprafata 3D devine o suprafata plana



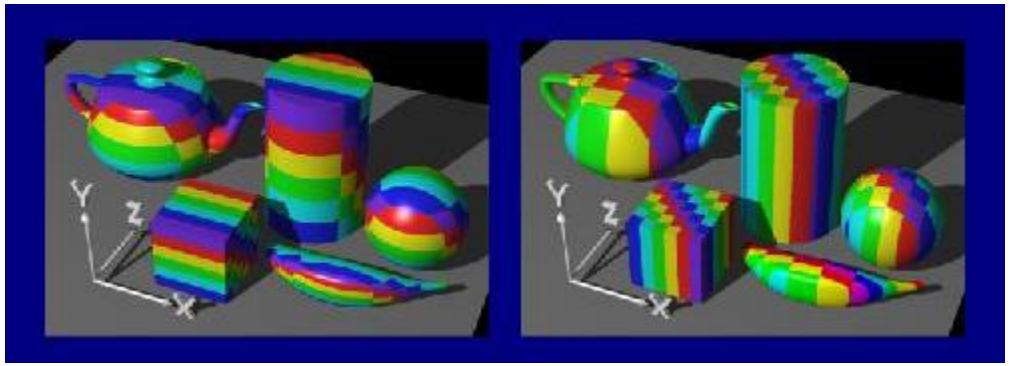
http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_2.htm

Exemple de mapari plane



Maparea plana prin renuntarea la coordonata z: toate punctele suprafetei cu aceleasi coordonate (x,y) au aceeasi culoare.

http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_2.htm

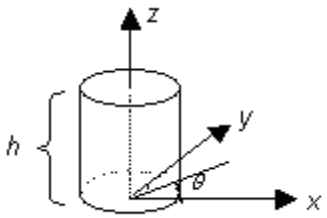


Maparea plana prin renuntarea la coordonata x, respectiv y.

Functii de mapare standard(2)

2. Maparea cilindrica

❖ Stabileste corespondenta dintre coordonatele (x,y,z) ale unui punct de pe o suprafata cilindrica si spatiul parametric al texturii, (u, v)

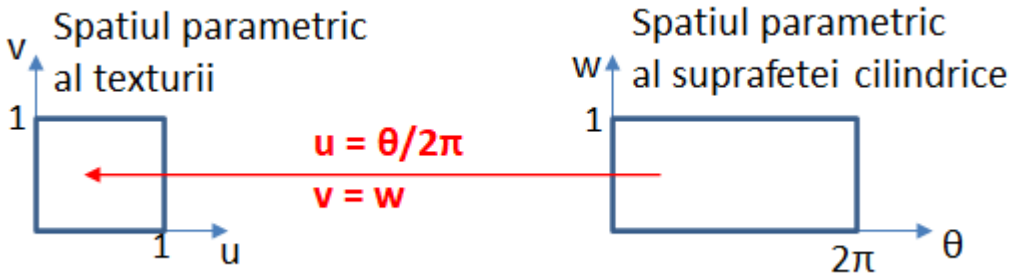


Ecuatiile parametrice ale unei suprafete cilindrice:

x=r*cos(θ)
y=r*sin(θ) 0<= θ< 2π, 0 < = w <= 1
z=w*h

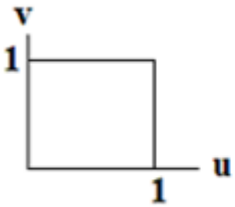
- Coordonatele cilindrice (*θ, w*) ale punctului (x, y, z)

θ = arctg(y/x)
w = z/h

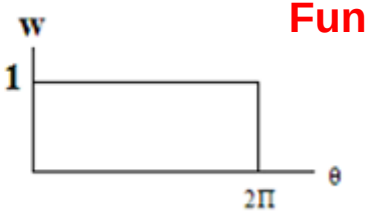


Functii de mapare standard (3)

$\theta = \mathrm{arctg}(y/x)$
$w = z/h$



$u = \theta/2\Pi$; $v=w$

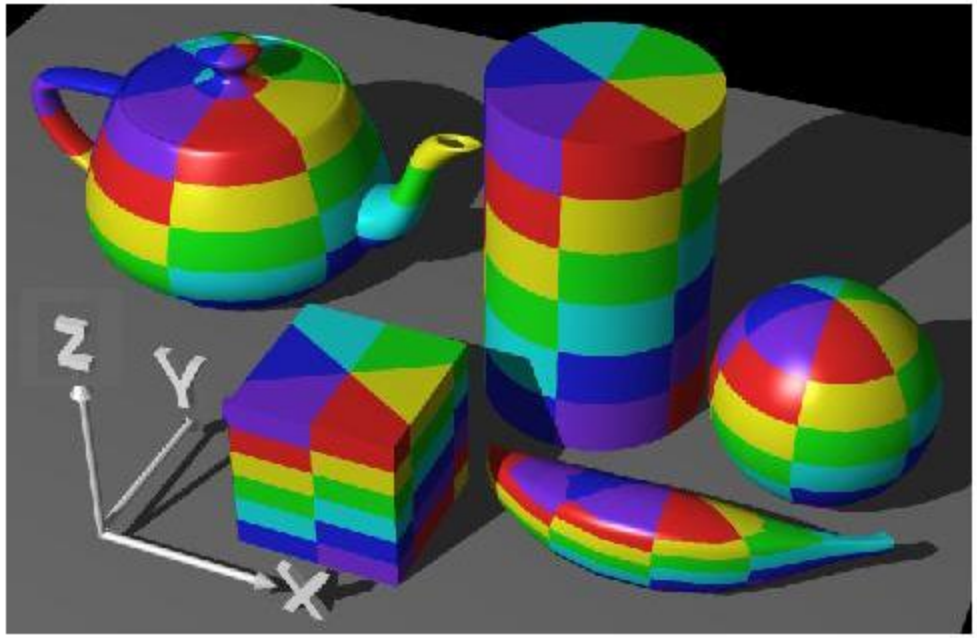


Functiile de mapare cilindrica

$u = (1/2\Pi)\mathrm{arctg}(y/x)$
$v = z/h$

Textura este “înfaşurată” în jurul obiectului

Transformarea patratelor texturii in punctele suprafetei de z= zmin si z= zmax



Functii de mapare standard(4)

3. Maparea sferica

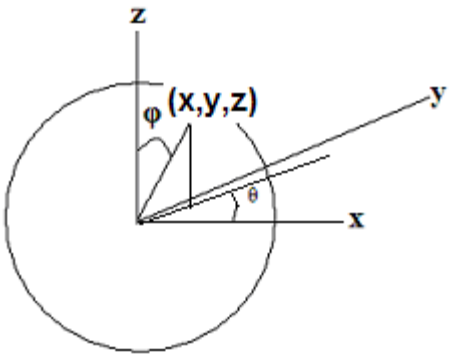
❖ Stabileste corespondenta dintre coordonatele (x,y,z) ale unui punct de pe o suprafata sferica si spatiul parametric al texturii, (u, v)

- Ecuatiile parametrice ale sferei:

$x = r*\cos(\theta)*\sin(\varphi) \qquad 0 <= \theta < 2\pi$

$y = r*\sin(\theta)*\sin(\varphi) \qquad 0 <= \varphi <= \pi$

$z = r*\cos(\varphi)$



- Coordonatele sferice ale punctului (x,y,z):

$\theta = \mathrm{arctg}(y/x)$

$\varphi = \arccos(z/r)$

Punand in corespondenta spatiul (u,v) spatiului parametric al sferei rezulta:

$u = \theta/2\pi$

$v = \varphi/\pi$

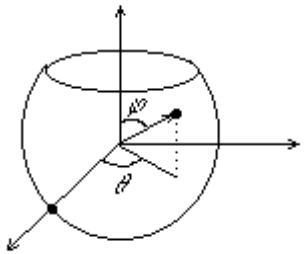
sau

$$\begin{cases} u = \dfrac{1}{2\pi} arctg\left(\dfrac{y}{x}\right) \\ v = \dfrac{1}{\pi}\arccos\left(\dfrac{z}{r}\right) = \dfrac{1}{\pi}\arccos\dfrac{z}{\sqrt{x^2+y^2+z^2}} \end{cases}$$

Functiile de mapare sferica

Functii de mapare standard (5)

Maparea pe un sector de sfera



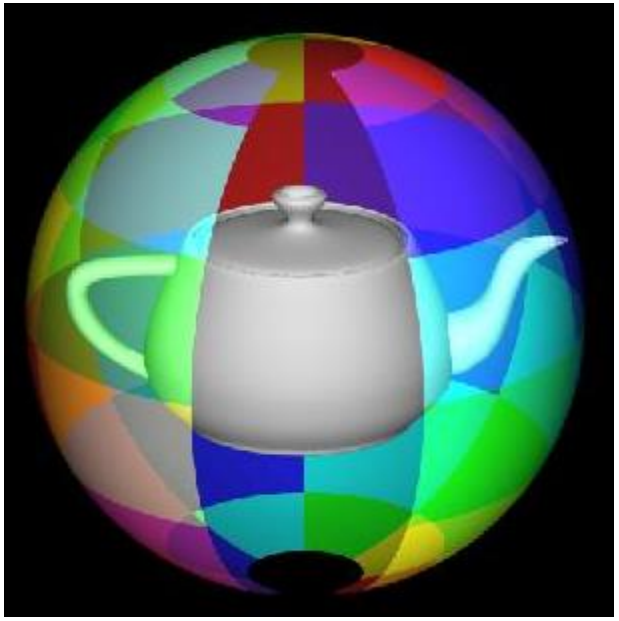
$0 <= \theta <= \pi/2$
$\pi/4 <= \varphi <= \pi/2$

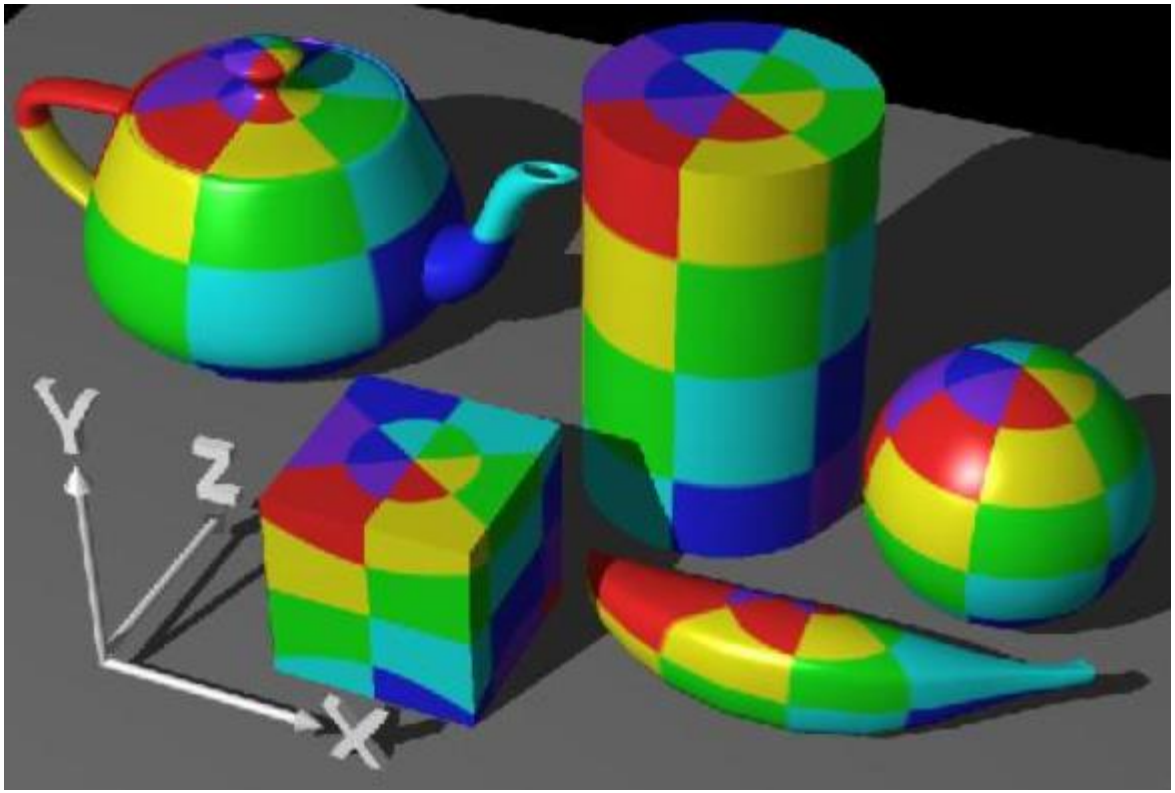
Functiile de mapare

$u = \theta/(\pi/2)$
$v = ((\pi/2) - \varphi) / (\pi/4)$

Exemple de mapari sferice

Maparea sferica “întinde” patratele texturii la nivelul ecuatorului si le “comprimă” proportional cu aproprierea de poli.



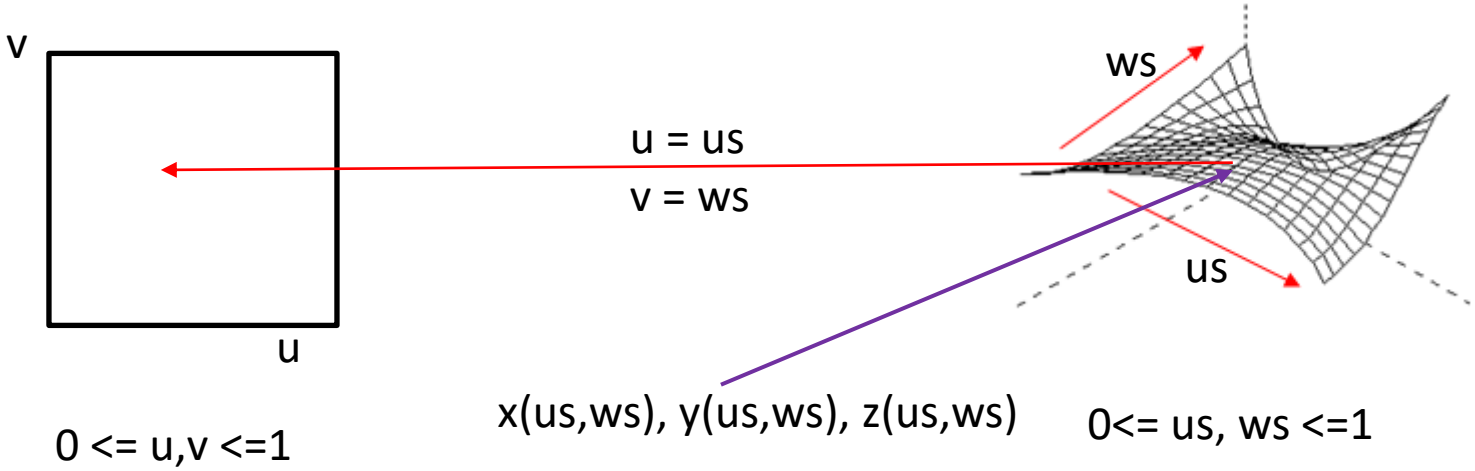


http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_2.htm

Aplicarea unei texturi pe o suprafata de forma libera

Orice suprafata de forma libera este definita intr-un spatiu parametric.

Un petic de suprafata de forma libera: p(us,ws) = [x(us,ws) y(us,ws) z(us,ws)], 0<= us, ws <=1

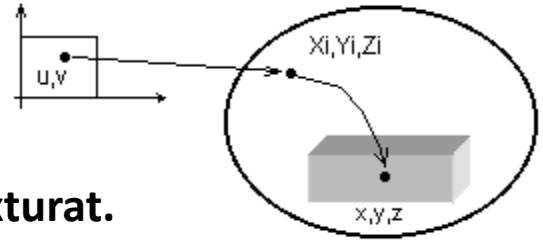


Maparea în 2 pasi (1)

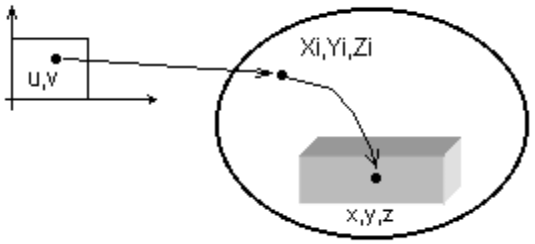
- Este o tehnica de mapare independenta de forma obiectului pe care se aplica textura. Metoda a fost definita de Bier si Sloan (1986).
- Pentru calculul coordonatelor textura ale punctelor suprafetei texturate se foloseste o suprafata intermediara pentru care exista o functie de mapare inversa.
- Aplicarea texturii este descompusa in doua operatii,

  al caror efect este :

  - **mularea texturii pe suprafata intermediara;**
  - **mularea suprafetei intermediare pe suprafata de texturat.**



Maparea în 2 pasi(2)



a) **maparea S (mapare inversa):**
u = fu(xi, yi, zi)
v = fv(xi, yi, zi)

b) (xi, yi, zi) ← **maparea O** ← (x, y, z)

a) Mularea texturii pe suprafata intermediara se realizeaza prin functia de mapare inversa. Aceasta este numita « **mapare S** », adica mapare pe *suprafata* intermediara.

b) Mularea suprafetei intermediare pe suprafata de texturat este numita « **mapare O** ». Ea trebuie sa puna in corespondenta fiecarui punct (x,y,z) al suprafetei (*obiectului*) de texturat, un punct (xi, yi, zi) al suprafetei intermediare

Maparea în 2 pasi(3)

Maparea S

Se folosesc 4 tipuri de suprafete intermediare :

1. un plan cu o orientare oarecare;
2. sfera;
3. cilindrul;
4. fețele unui cub.

Alegerea tipului suprafetei intermediare este dependenta de forma obiectului pe care trebuie sa fie aplicata textura.

Maparea O

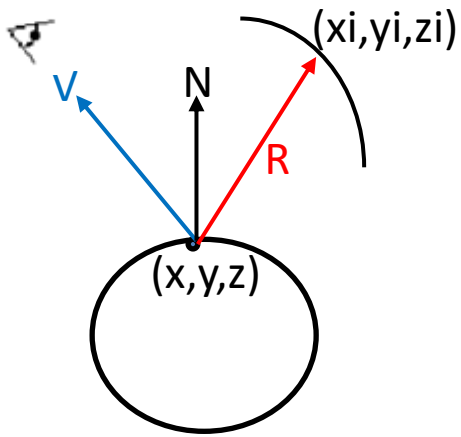
Autorii metodei au definit 4 tipuri de mapari O:

Maparea în 2 pasi(4)

Mapari O

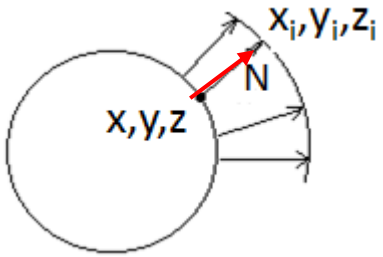
(x,y,z) – un punct al suprafetei , pentru care se doreste calculul coordonatelor (u,v)

(xi, yi, zi) – punctul corespunzator lui (x,y,z) pe suprafata intermediara



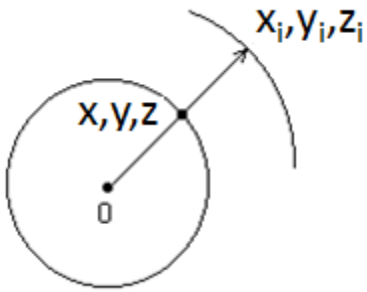
1. (xi, yi, zi), se obtine intersectand suprafata intermediara cu vectorul reflectat al vederii in punctul (x,y,z). R este simetric cu V fata de N.

Metoda este dependenta de pozitia observatorului (se foloseste la « maparea mediului inconjurator »)

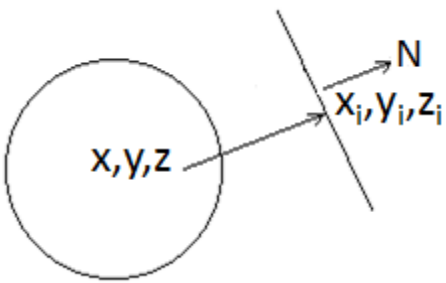


2. (xi,yi,zi) este intersectia suprafetei intermediare cu normala la obiect în (x,y,z).

Maparea în 2 pasi(5)



3. (xi,yi,zi) este intersectia suprafetei intermediare cu dreapta care trece prin (x,y,z) si centroidul obiectului.



4. (xi,yi,zi) este intersectia suprafetei intermediare cu dreapta care trece prin (x,y,z) si are directia normalei la suprafata intermediara.

Acest tip de mapare O este corelat cu o mapare S pe un plan sau pe fetele unui cub.

Combinatia « mapare S pe cilindru » - « mapare O de tip 4 » este numita « shrinkwrap ».

Aplicarea texturilor pe fetele unei retele poligonale 3D (1)

In sistemele grafice actuale, orice suprafata 3D este descompusa, in vederea redarii sale, într-o rețea de fațete triunghiulare: textura se aplica pe suprafetele triunghiulare la momentul rasterizarii lor.

Pentru fiecare fragment rezultat din rasterizare se poate obtine o culoare din textură in 2 moduri:

1. In fragment shader, se calculeaza coordonatele (uf,vf) corespunzatoare adresei fragmentului în spatiul obiect (xf,yf,zf), folosind functii de mapare standard

   uf=fu(xf,yf,zf), vf= fv(xf,yf,zf)

   sau maparea in 2 pasi.

2. In fragment shader, se utilizeaza coordonatele (uf,vf) calculate de rasterizator prin interpolarea coordonatelor (ui,vi) atasate varfurilor primitivei din care face parte fragmentul.

   - Pentru rezultate corecte este necesara efectuarea unei interpolari perspectiva. Interpolarea liniara este mai rapida dar poate produce defecte.

❖ Metode mai exacte (ex. Mip-mapping) calculeaza coordonatele textura ale unui fragment tinand cont de aria suprafetei textură comprimată in fragment.

Aplicarea texturilor pe fetele unei retele poligonale 3D(2)

- In cazul utilizarii mai multor texturi la redarea unei primitive, in fragment shader se extrage cate o culoare din fiecare textura, fie calculand adresele (ui,vi) in fragment shader, fie folosind coordonatele textura interpolate. Culorile extrase din diferite texturi pot fi combinate in diferite moduri pentru a se obtine culoarea fragmentului.

- Daca o imagine textura trebuie mulata pe intreaga retea de fatete care aproximeaza o suprafata, atunci varfurile retelei trebuie sa se mapeze pe intreg spatiul 0 <= (u,v) <= 1.
- Coordonatele (ui,vi) atasate varfurilor unei primitive se pot obtine intr-o aplicatie OpenGL printr-o mapare in 2 pasi sau folosind functii de mapare inversa.
- Aplicatiile de modelare a suprafetelor 3D permit atasarea de coordonate textura varfurilor suprafetei. Obiectele importate contin coordonatele (u,v) atasate varfurilor obiectului.
- Coordonatele textura ale varfurilor se transmit ca intrari la Vertex shader.

Transparenta texturii (1)

- Imaginea textura poate avea o componenta aditionala, numita “canalul alfa”, prin care se poate modela transparenta texturii.
- Fiecare texel este in acest caz reprezentat prin 4 componente: (R,G,B,A), unde A reprezinta opacitatea.
  - daca A = 1, atunci texelul este complet opac
  - daca A=0, texelul este complet transparent
- Culoarea unui fragment se poate obtine: dintr-o textura, combinand culori din mai multe texturi, combinand culoarea fragmentului calculata pe baza geometriei (modelul Gouraud sau Phong) cu o culoare extrasa dintr-o textura (sau mai multe).
- Atunci cand culoarea fragmentului are opacitatea $0 < A < 1$, se poate combina culoarea sa cu aceea a pixelului destinatie, printr-o operatie de “amestec” între cele 2 culori, specificata prin functia glBlendFunc(..).

Transparenta texturii (2)

glEnable(GL_BLEND);

Culoarea rezultata din amestec:

Crezultat = Csource*Fsource + Cdestination*Fdestination

Csource – culoarea fragmentului (obtinuta din textura);

Cdestination – culoarea pixelului în care se va afişa fragmentul

Fsource – factor care determina influenta valorii alfa asupra culorii Csource

Fdestination - factor care determina influenta valorii alfa asupra culorii Cdestination

Fsource şi Fdestination pot fi specificati astfel:

GlBlendFunc(GlEnum Fsource, GlEnum Fdestination);

Exemplu:

glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

Crezultat = Csource* Asource + Cdestination* (1 – Asource)

Valorile posibile pentru Fsource si Fdestination sunt cuprinse in tabelul urmator:

Transparenta texturii (3)

GL_ZERO	Factor is equal to 0.
GL_ONE	Factor is equal to 1.
GL_SRC_COLOR	Factor is equal to the source color vector $\bar{C}_{source}$.
GL_ONE_MINUS_SRC_COLOR	Factor is equal to 1 minus the source color vector: $1 - \bar{C}_{source}$.
GL_DST_COLOR	Factor is equal to the destination color vector $\bar{C}_{destination}$
GL_ONE_MINUS_DST_COLOR	Factor is equal to 1 minus the destination color vector: $1 - \bar{C}_{destination}$.
GL_SRC_ALPHA	Factor is equal to the $alpha$ component of the source color vector $\bar{C}_{source}$.
GL_ONE_MINUS_SRC_ALPHA	Factor is equal to $1 - alpha$ of the source color vector $\bar{C}_{source}$.
GL_DST_ALPHA	Factor is equal to the $alpha$ component of the destination color vector $\bar{C}_{destination}$.
GL_ONE_MINUS_DST_ALPHA	Factor is equal to $1 - alpha$ of the destination color vector $\bar{C}_{destination}$.
GL_CONSTANT_COLOR	Factor is equal to the constant color vector $\bar{C}_{constant}$.
GL_ONE_MINUS_CONSTANT_COLOR	Factor is equal to 1 - the constant color vector $\bar{C}_{constant}$.
GL_CONSTANT_ALPHA	Factor is equal to the $alpha$ component of the constant color vector $\bar{C}_{constant}$.
GL_ONE_MINUS_CONSTANT_ALPHA	Factor is equal to $1 - alpha$ of the constant color vector $\bar{C}_{constant}$.

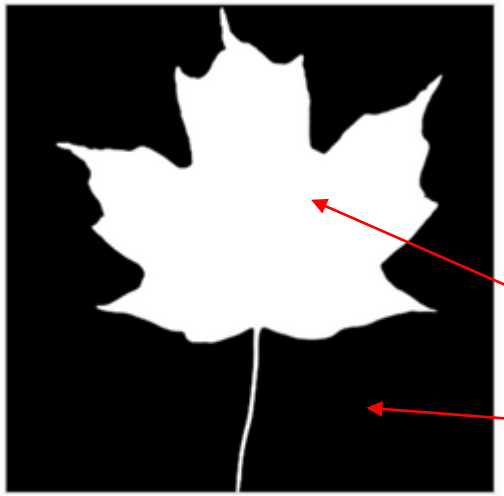
Detalii la: https://learnopengl.com/Advanced-OpenGL/Blending

Transparenta texturii (4)

- Atunci cand culoarea fragmentului are opacitatea A=0 sau sub o anumita valoare, se poate cere eliminarea sa (suprascrierea unui pixel cu ceva transparent nu are efect).

```
uniform sampler2D texture1;
in vec2 TexCoords;
out vec4 FragColor;
//Fragment shader
void main() {
   vec4 texColor = texture(texture1, TexCoords);
   if(texColor.a < 0.1) discard; // fragmentul este eliminat – nu va fi procesat in continuare
   FragColor = texColor;
}
```

Transparenta texturii (5)



Imaginea textura

Desenul frunzei este memorat pe canalul alfa al texturii.

Toti texelii texturii au culoarea albastru.

(0,0,1,1) – texeli opaci

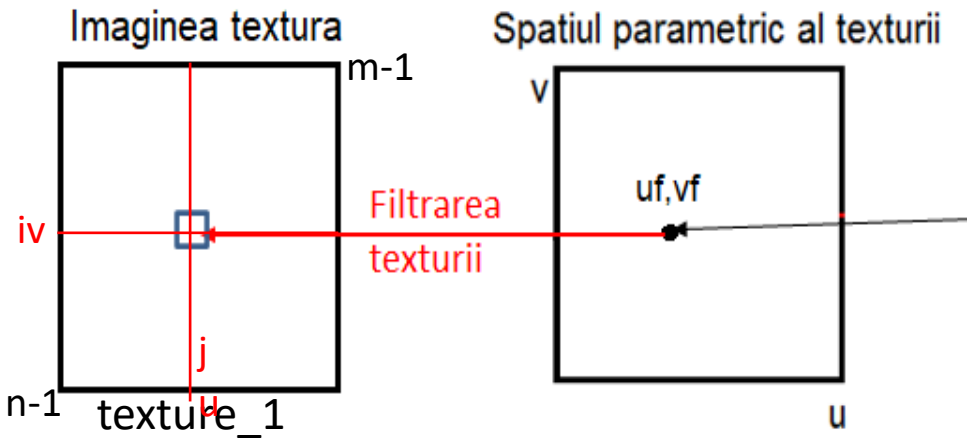
(0,0,1,0) – texeli transparenti

Imaginea texturii aplicată pe o suprafață albă.

```
void main() {
  vec4 texColor = texture(texture1, TexCoords);
 // pentru eficiență:
   if(texColor.a == 0) discard;
   FragColor = texColor;
 }
```

Filtrarea texturii (1)

```
//Fragment shader
uniform sampler2D texture_1;   // imaginea textura
in vec2 texcoord; //coordonatele textura – intrare pentru Fragment shader

layout(location = 0) out vec4 out_color;
void main()
{ vec4 color = texture2D(texture_1, texcoord);
out_color = color;
}
```



texture2D(texture_1, texcoord)

uf = texcoord.x
vf = texcoord.y

Adresa in spatiul texturii corespunzatoare coordonatelor (uf,vf) sunt 2 numere reale:

ju = uf*(m-1) iv = (1-vf)*(n-1)

Filtrarea texturii (2)

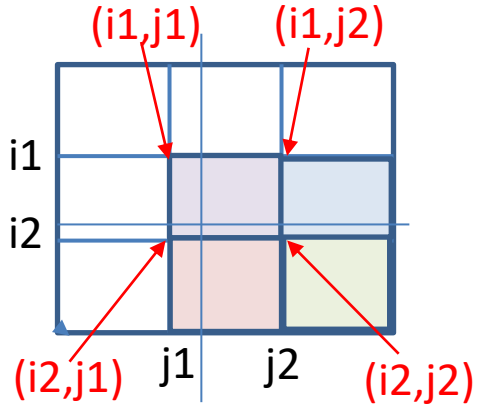
- Calculul culorii dintr-o textura folosind coordonatele (uf,vf), se numeste “filtrarea texturii”.
- (iv, ju) – coordonatele reale textura corespunzatoare adresei (uf,uv) : i1 < iv < i2, j1 < ju < j2
- [i1][j1], [i1][j2], [i2][j1], [i2][j2] sunt adrese in spatiul discret al texturii (adrese de texeli)

Cazul 0<= uf, vf <= 1

Sunt doua moduri standard de filtrare:

1) color = textura [i][j], unde [i][j] este adresa cea mai apropiata de (iv,ju) in spatiul texturii

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER/GL_TEXTURE_MIN_FILTER, GL_NEAREST);



i1 = 12, i2= 13,
j1 = 4, j2 = 5
iv = 12.8, ju = 4.3
Culoare_textura ← textura[i2][j1]

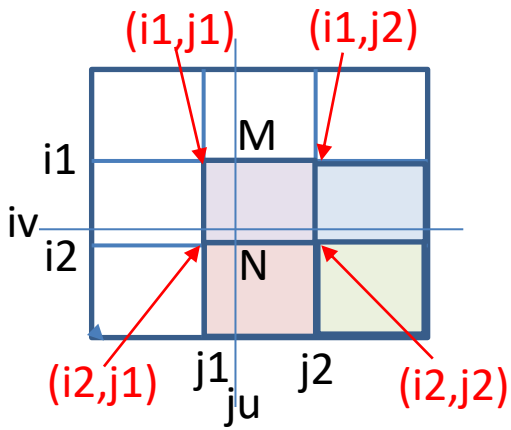
— rapida dar poate produce defecte in imagine

— este metoda de filtrare implicita

Filtrarea texturii (3)

2) color = interpolare biliniara intre culorile texelilor de adrese [i1][j1], [i1][j2], [i2][j1], [i2][j2]

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER/GL_TEXTURE_MIN_FILTER, GL_LINEAR);



Culoare_textura ← interpolare intre culorile texelilor de adrese [i1][j1], [i1][j2], [i2][j1], [i2][j2]

- scade viteza de calcul a culorii fragmentului
- poate produce un efect de înceţoşare

Culoarea extrasa din matricea textura se calculeaza astfel:

M(i1,ju); N(i2,ju);

C(i,j) – culoarea texelului de indici (i,j)

Cul(M) = (j2-ju)*C(i1,j1) + (ju-j1)*C(i1,j2)

Cul(N) = (j2-ju)*C(i2,j1) + (ju-j1)*C(i2,j2)

Culoare_textura = (i2-iv)*Cul(M) + (iv-i1)*Cul(N)

Filtrarea texturii (4)

- La momentul rasterizării unei suprafețe pe care se aplică o textură, GPU trebuie să acceseze textura pentru a extrage culoarea pentru un fragment.
- Textura trebuie sa fie mărită sau micşorată în funcție de dimensiunea în spațiul ecran a suprafeței.
- **Mărire (magnification): un texel contribuie la culoarea mai multor pixeli adiacenți**

  Exemplu, atunci când un jucator se apropie foarte mult de un perete, numărul de pixeli care trebuie sa fie acoperiți de textură este mai mare decat numarul de texeli ai texturii: textura 256x256 texeli trebuie aplicata pe o suprafata de 512x512 pixeli.
- **Micşorare (minification): mai mulți texeli contribuie la culoarea unui pixel**

  Exemplu, atunci când jucatorul se îndepărtează foarte mult de perete, suprafața peretelui acoperă un număr de pixeli mult mai mic decat numărul de texeli ai texturii: textura este de 256x256 texeli şi trebuie afişată pe o suprafață de 8x8 pixeli.

Filtrarea texturii (5)

- OpenGL dă posibilitatea să se specifice modul de accesare a texturii în cele 2 cazuri:

  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER /GL_TEXTURE_MIN_FILTER, GL_NEAREST/ GL_LINEAR);

- **În cazul măririi**, filtrarea NEAREST nu dă rezultate bune, dar filtrarea LINEAR dă rezultate satisfacatoare.
- **În cazul micşorării:**

- Presupunând că textura este de 256x256 texeli şi trebuie afişată pe o suprafață de 8x8 pixeli, ar trebui ca 1024 texeli sa fie „comprimați” pentru a se obține culoarea unui pixel.
- Filtrarea NEAREST are ca efect colorarea a 2 pixeli adiacenți folosind culorile a 2 texeli (foarte) îndepărtați între ei în spațiul texturii – eşantionare rară a texturii. Numai 64 de texeli din cei 256x256 sunt folositi la afişarea suprafeței.
- Filtrarea LINEAR nu are sens, deoarece conduce la interpolarea între culorile a 4 texeli îndepărtați între ei în spațiul texturii (eşantionare rară a texturii)

Filtrarea texturii (6)

- Ambele moduri de filtrare produc imagini cu defecte, iar la schimbarea direcției observatorului apar pâlpâiri ale imaginii din cauza adresării aleatorii a texturii → Exemplu curs
- Totodată, accesarea aleatorie a texturii pentru pixeli adiacenți este ineficientă: creşte timpul necesar producerii imaginii.

Soluția: metoda mip-mapping, care foloseste pentru filtrarea texturii un set de texturi pre-comprimate.

- Ideea mip-mapping: **utilizarea de imagini textura de rezolutii diferite în funcție de mărimea în imagine a suprafeței texturate**.
- Pornind de la o imagine textură, se pot obtine texturi de rezolutie mai joasa prin medierea culorilor din textura initiala.
- "MIP“: acronim al frazei din limba latina: *multum in parvo*, insemnand “mult in puțin"

Redarea suprafetelor 3D folosind texturi - 2

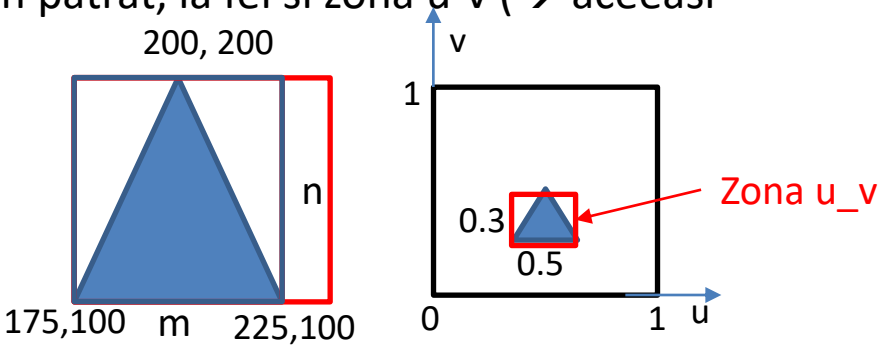
Prof. univ. dr. ing. Florica Moldoveanu

Metoda "mip-mapping" (1)

- Necesară atunci când textura trebuie să fie micşorată.
- Textura redata intr-un pixel trebuie sa fie comprimata in functie de dimensiunea primitivei texturate in spatiul ecran (distanța de la observator la suprafața texturata).
- Factorul de compresie a texturii intr-un pixel se calculeaza pe GPU pentru fiecare primitiva grafica, dupa transformarea varfurilor în spatiul ecran, tinand cont de suprafata primitivei în spatiul ecran si zona u-v folosita la texturarea sa.
- Suprafata primitivei se aproximeaza printr-un patrat, la fel si zona u-v (→ aceeasi compresie pe axele u si v)

Exemplu: m=50, n= 100

se aprox printr-un patrat cu latura de 100



Zona u_v se aproximeaza prin patratul incadrator cu latura de 0.5

Factorul de compresie a texturii într-un pixel acoperit de primitivă: f=0.5/100 = 1/200

Daca rezolutia texturii este de 512x512 texeli → 512/200 = 2.56

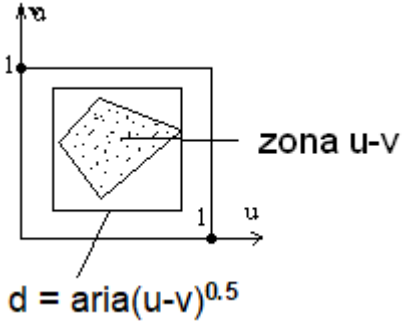
→ 2.56x2.56 = 6.55 texeli se comprima într-un pixel.

Metoda "mip-mapping" (2)

- In calculul culorii unui fragment ar trebui sa se foloseasca o textura comprimata cu factorul de compresie calculat pt. primitiva din care face parte >> un nr. f mare de texturi comprimate.
- Se calculeaza o piramida de imagini textură comprimate, fiecare nivel al piramidei corespunzand unui factor de compresie al texturii → nr de factori de compresie este limitat la nr. de niveluri ale piramidei!
  - De exemplu, fiecare imagine textura comprimata se obtine prin aplicarea unui filtru trece-jos, într-o fereastra de 2x2 pixeli, pe imaginea textura de pe nivelul anterior.
- Culoarea unui fragment se obtine din imaginile textura ai caror factori de compresie sunt cei mai apropiati de cel calculat pentru primitiva din care face parte fragmentul.
  - Fie aria(u-v): aria suprafetei din spatiul (u-v) corespunzatoare unui texel

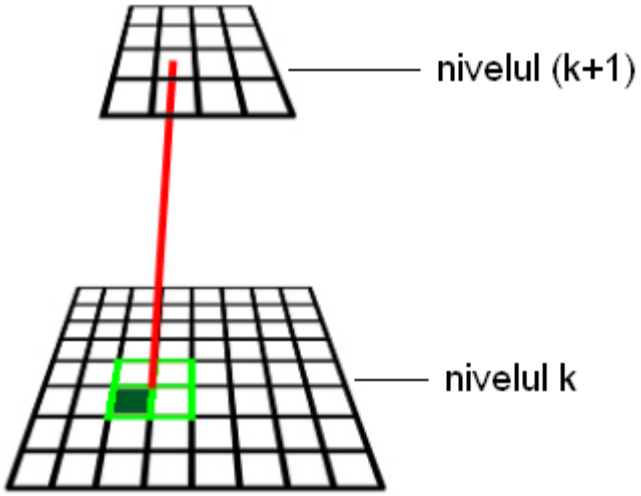
    Pentru o textura de nxn texeli, aria u-v = (1/n)x(1/n)

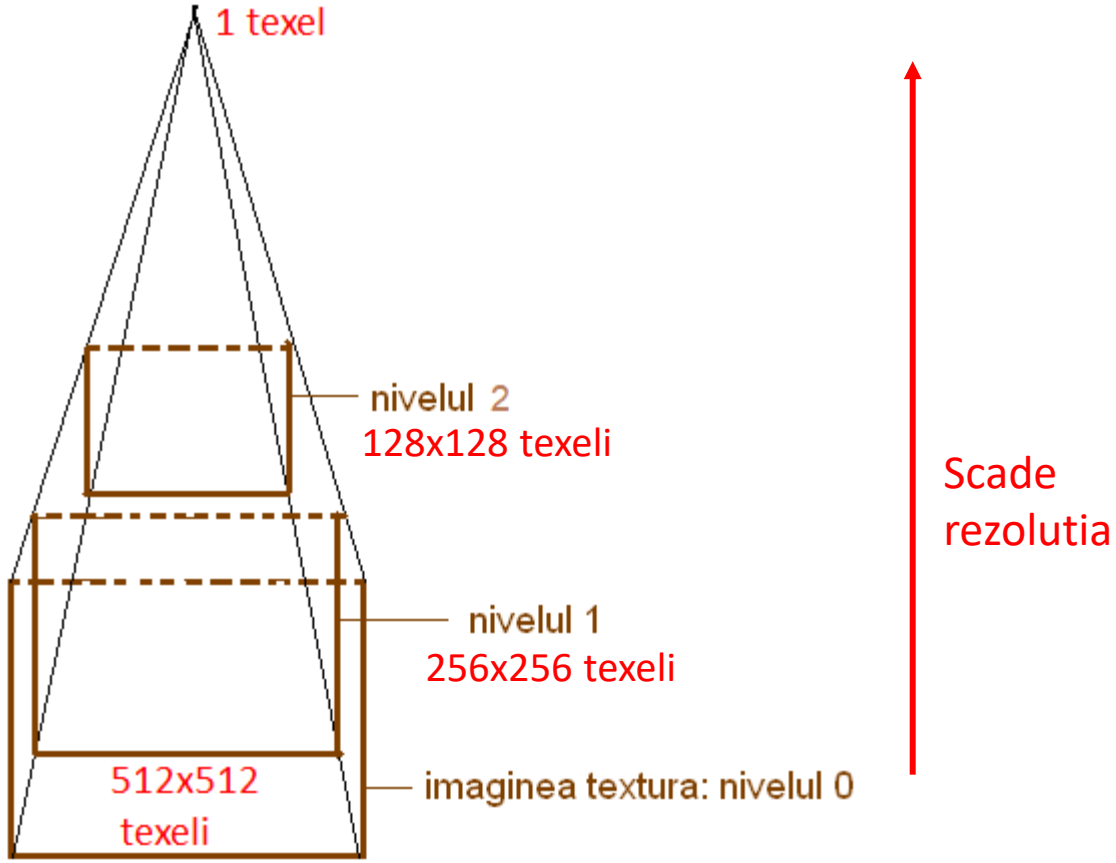
    Pentru n=1 → aria (u-v) = 1 → d=1: compresie maximă (întreaga textură comprimată într-un texel)



- d reprezinta factorul de compresie al texturii

Metoda "mip-mapping" (3)
- Piramida mip-map -



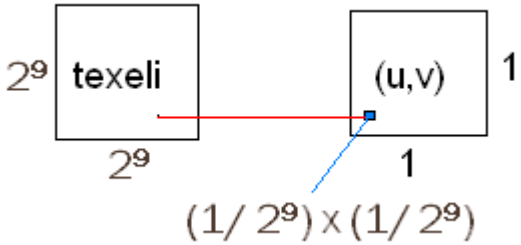


La momentul rasterizarii unei primitive se alege imaginea textura de pe nivelul cu rezolutia cea mai apropiata de dimensiunea in imagine a primitivei.

Metoda "mip-mapping" (4)
- Calculul piramidei mip-map -

- Pentru o imagine textura de 512x512 texeli rezulta o piramida compusa din 10 niveluri:
- Nivelul 0: intreaga imagine textura (512x512)= $\mathbf{2^9 \times 2^9}$

  aria (u-v) coresp unui texel = $\mathbf{(1/\ 2^9) \times (1/\ 2^9)} = 1/2^{18} = \mathbf{2^{2(0-9)}}$



- Nivelul 1: 256x256 texeli = $2^8 \times 2^8$ →aria (u-v) coresp unui texel

  $=(1/2^8)\times(1/2^8) = (2/\ 2^9) \times (2/\ 2^9) = \mathbf{2^{2(1-9)}}$
- Nivelul 2: 128x128 texeli = $2^7 \times 2^7$ →aria (u-v) coresp unui texel = $(4/\ 2^9) \times (4/\ 2^9) = \mathbf{2^{2(2-9)}}$
- Nivelul k: aria (u-v) coresp unui texel = $\mathbf{2^{2(k-9)}}$
- Nivelul 9: 1 texel →aria (u-v) coresp unui texel = 1

  (compresie maxima: intreaga imagine textura este comprimata intr-un singur texel)
- Factorul de compresie al imaginii textura de pe nivelul k este:

  $d_k = (\text{aria } (u\text{-}v)_k)^{0.5} = \mathbf{2^{(k-9)}}$
- Factorul de compresie al imaginii textura de pe nivelul k+1 este:

  $d_{k+1} = (\text{aria } (u\text{-}v)_{k+1})^{0.5} = \mathbf{2^{(k-8)}}$

Metoda "mip-mapping" (5)
– filtrarea texturii -

Pentru calculul culorii unui fragment, se adreseaza piramida mip-map prin 3 coordonate: (u, v, d):

d, care se foloseste pentru a selecta nivelul piramidei din care se calculeaza culoarea sau nivelurile intre care se va interpola pentru calculul culorii; **d poate fi considerata ca fiind coordonata verticala a piramidei**;

(u,v) prin care se calculeaza o culoare prin filtrare de pe un nivel al piramidei.

- Fie uf si vf coordonatele folosite pentru calculul culorii unui fragment
- Fie df factorul de compresie al texturii intr-un pixel pentru primitiva din care face parte fragmentul.

Daca nu s-a cerut interpolare intre nivelurile mip-map

glTexParameteri(….GL_NEAREST_MIPMAP_NEAREST sau GL_LINEAR_MIPMAP_NEAREST)

- se determina nivelul piramidei pentru care factorul de compresie este cel mai apropiat de df.

- se calculeaza o culoare din textura corespunzatoare nivelului folosind (uf,vf)

Metoda "mip-mapping" (6)
– filtrarea texturii -

Daca s-a cerut interpolare între nivelurile mip-map:

glTexParameteri(….GL_NEAREST_MIPMAP_LINEAR sau GL_LINEAR_MIPMAP_LINEAR)

Fie: $\mathbf{d_k < df < d_{k+1}}$

- Folosind (uf, vf)

  - se determina o **culoare din textura de pe nivelul k**, $\mathbf{C_k}$

  - se determina o **culoare din textura de pe nivelul k+1**, $\mathbf{C_{k+1}}$

- Se determina culoarea de afisare a fragmentului prin interpolare liniara intre C_k si C_{k+1}**:**

$$\mathbf{C = C_k + dInterp\ (C_{k+1} - C_k)}$$

unde **dInterp** se determina din conditia: $df = d_k + \mathbf{dInterp}\ (d_{k+1} - d_k) = \mathbf{2^{k-9}} + dInterp\ (2^{\mathbf{k-8}} - 2^{\mathbf{k-9}})$

Rezulta: $\mathbf{dInterp} = 2^{9-k} * df - 1$

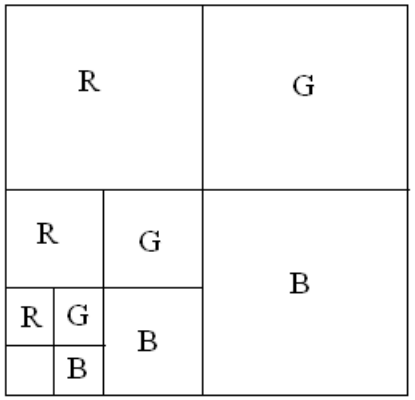
Exemplu:

Factorul de compresie: df=1/40 → d3 (1/64) < df < d4 (1/32); $dInterp = 2^6 * 1/40 - 1 = 0.6$

$C = C_3 + 0.6\ (C_4 - C_3) = C_3\ (1-0.6) + 0.6 * C_4 = C_3 * 0.4 + C_4 * 0.6$

Metoda "mip-mapping" (7)
- memorarea piramidei mip-map -

Pentru o imagine textura color de 512x512 texeli, piramida se poate memora pe (1024 x1024) octeti, cu fiecare canal de culoare pe un plan separat, astfel :



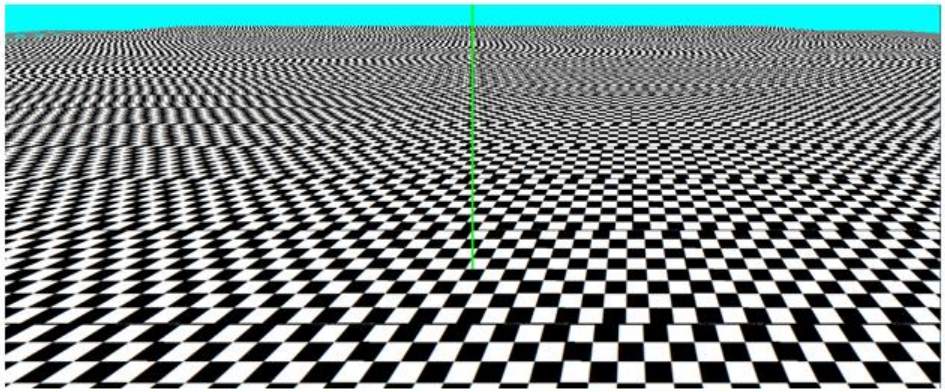
- Piramida poate fi vizualizata ca un patrat de 4 ori mai mare decat dimensiunile texturii originale.
- Fiecare plan necesita 1/3 din spatiul de memorare pentru un nivel al piramidei, deci intreaga piramida necesita 4/3 din spatiul de memorare al texturii initiale, adica un consum de memorie suplimentar de 1/3 ≈ 33% .

Piramida mip-map poate fi generata in aplicatia OpenGl si transmisa la GPU:

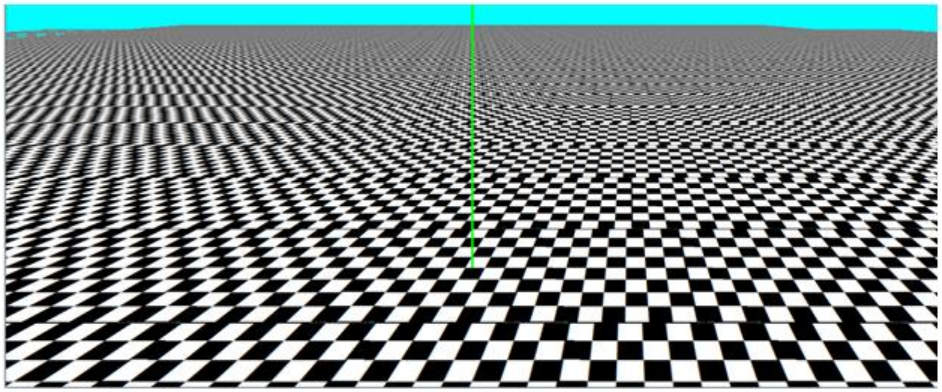
```
glGenerateMipmap(GL_TEXTURE_2D);
```

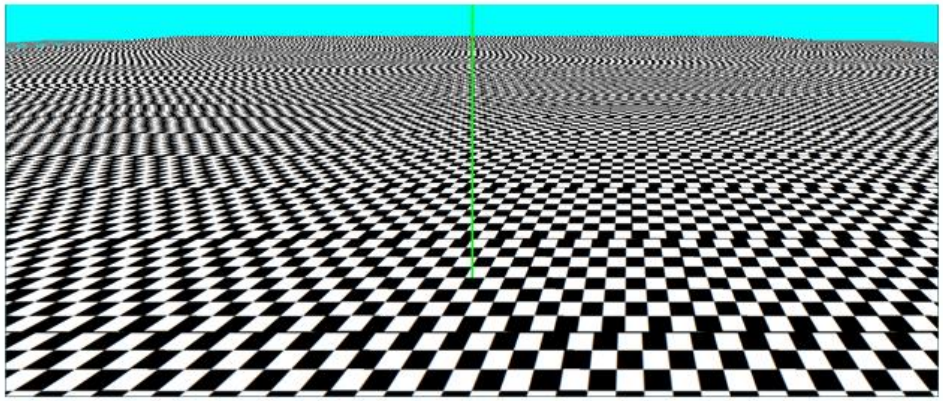
Metoda "mip-mapping"
- Capturi ecran-

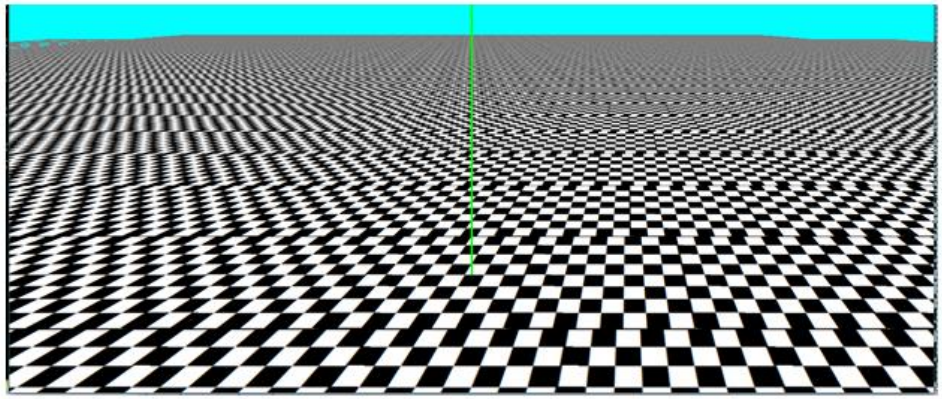
Fără mip-mapping



Cu mip-mapping







Metoda "mip-mapping" (8)
- Aprecieri -

- **Tehnica mip-mapping:**

  - da rezultate bune, chiar daca se bazeaza pe o serie de aproximari

  - este implementata de GPU: trebuie sa i se transmita imaginile textura ale piramidei

- **Existenta piramidei de imagini simplifica filtrarea texturii in cazul micsorarii**: pe masura ce suprafata se indeparteaza de observator, culoarea se calculeaza din nivelurile din ce in ce mai mari ale piramidei.

- **Avantajelei metodei:**

➢ Elimina artefactele vizuale produse de accesarea aleatorie a texturii la micsorarea suprafetei texturate.

➢ Creste performanta procesului de extragere a culorilor din textura, atunci cand un numar mare de texeli trebuie folositi pentru un pixel.

➢ Consumul de memorie suplimentar pentru memorarea piramidei este de 33%.

Comprimare şi repetare (1)

Coordonate (u, v) in afara domeniului [0,1]

▪Coordonatele (ui, vi) asociate varfurilor pot fi in afara domeniului [0,1].

➢ Pot rezulta coordonate (u, v) la nivel de fragment în afara domeniului [0,1].

▪ Coordonatele (u, v) în afara domeniului [0,1] rezultate in procesul de rasterizare sunt transformate de GPU în coordonate 0<= uf, vf <=1.

▪ Modul de transformare a coordonatelor (u,v) din afara domeniului [0,1] se specifica separat pe fiecare axa: u/v.

▪ Exista cateva moduri standard de transformare a coordonatelor textura in domeniul [0,1]. Doua dintre acestea sunt: “clamping” si “wrapping (sau “repeating).

Clamping (Comprimare)

Coordonatele u/v sunt transformate in uf/ vf astfel:

uf/ vf = min(max(0,u),1)/ min(max(0,v),1)

Deci: daca u / v <0, atunci uf / vf =0; daca u / v >1, atunci uf / vf=1

daca 0<= u, v <=1, atunci uf, vf = u, v.

Comprimare şi repetare (2)

Clamping (Comprimare)

Aplicarea acestei transformari are ca efect propagarea culorii de la marginea texturii de-a lungul directiei in care coordonata textura este < 0 sau > 1.

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, *GL_CLAMP_TO_EDGE*);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, *GL_CLAMP_TO_EDGE*);

Comprimare şi repetare (3)

Wrapping (Repetare)

- Pentru accesarea texturii se calculeaza (uf,vf) cu formula:

  uf/vf = u- [u] / v-[v]

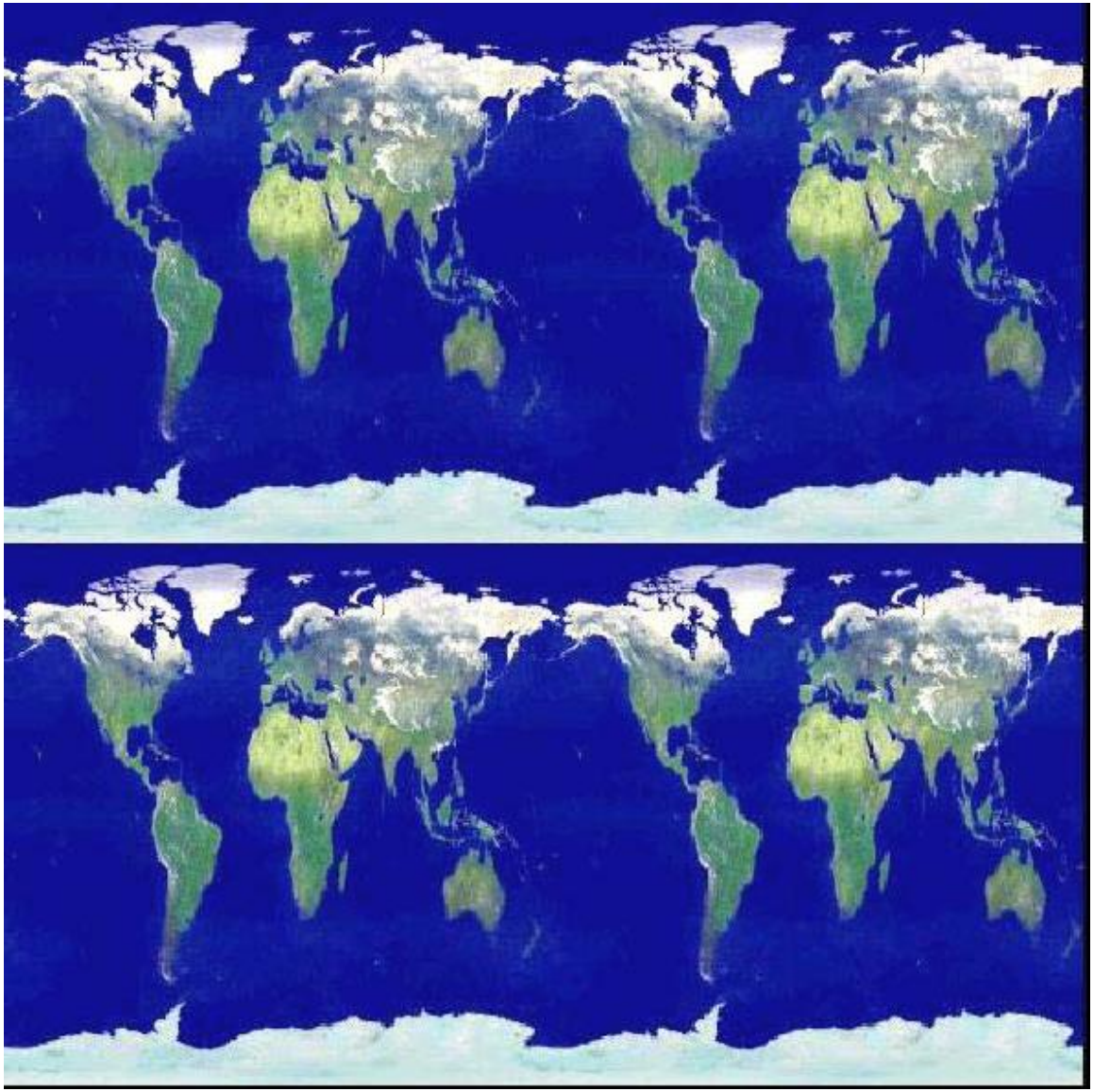
  unde [w] reprezinta cel mai mare intreg mai mic sau egal cu w.

  De exemplu, (-0.1, 1.5) se transforma in (-0.1 – (-1), 1.5 – (1)) = (0.9, 0.5).

- Acest mod de transformare permite repetarea unei texturi, producand un efect periodic.
- La utilizarea acestui mod este necesar ca laturile stanga/dreapta, respectiv sus/jos ale imaginii textura se se potriveasca, astfel incat sa nu se observe marginile texturii.
- Daca pentru una dintre coordonate se utilizeaza transformarea “comprimare” iar pentru cealalta “repetare”, textura se numeste “cilindrica”.
- Daca pentru ambele coordonate se utilizeaza modul “repetare”, textura se numeste “toroidala”.

Comprimare şi repetare (4)

Wrapping (Repetare)



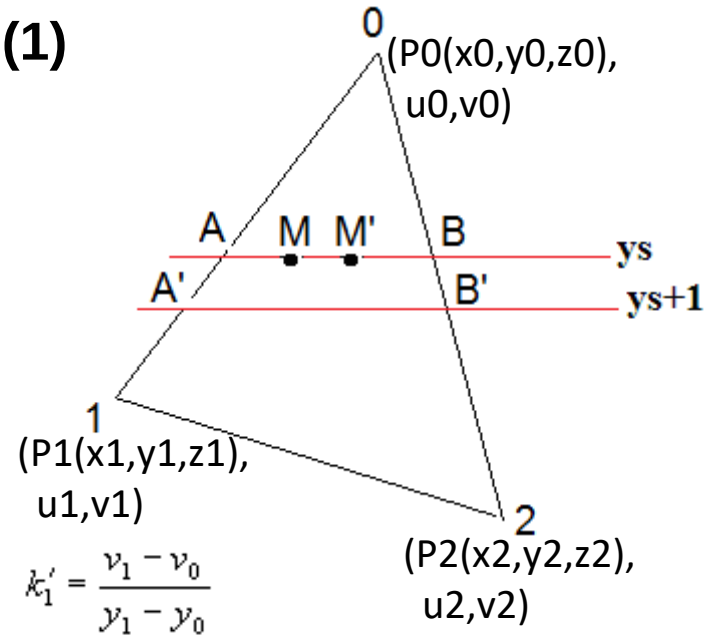
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

Interpolarea coordonatelor textura (1)

- Interpolarea este efectuata de GPU, intre coordonatele textura ale varfurilor unei primitive, pentru obtinerea coordonatelor textura ale unui fragment.
- In procesul de rasterizare, imaginea este calculata linie cu linie iar pentru eliminarea partilor nevizibile se foloseste algoritmul Z-buffer.
- Algoritmul de rasterizare primeste la intrare, pentru fiecare varf, coordonatele (x,y,z) dupa transformarea de vizualizare si coordonatele (u,v) din spatiul textura asociate varfului.
- In interpolarea liniara, calculul coordonatelor (u,v) pentru fragmentele interioare poligonului este similar calculului culorilor in modelul Gouraud:
  - Pentru fragmentele de pe laturi - prin interpolare liniara intre valorile asociate varfurilor
  - Pentru fiecare fragment interior – prin interpolare liniara intre valorile asociate extremitatilor segmentului de linie imagine interior poligonului

Interpolarea coordonatelor textura (2)

Interpolarea liniara (1)



$$y_s = y_A = y_0 + t(y_1 - y_0) \Rightarrow t_A = \frac{y_s - y_0}{y_1 - y_0}$$

$$u_A = u_0 + t_A(u_1 - u_0) = u_0 + (y_s - y_0)\frac{u_1 - u_0}{y_1 - y_0} = u_0 + k_1(y_s - y_0), k_1 = const.$$

$$y_s + 1 \Rightarrow u'_A = u_0 + k_1(y_s + 1 - y_0) = u_0 + k_1(y_s - y_0) + k_1 \Rightarrow \begin{cases} u'_A = u_A + k_1 \\ v'_A = v_A + k'_1 \end{cases}, \quad k'_1 = \frac{v_1 - v_0}{y_1 - y_0}$$

Analog in B: $\begin{cases} u'_B = u_B + k_2 \\ v'_B = v_B + k'_2 \end{cases}$

(u,v) pentru fragmentele interioare
se obtin prin calcule incrementale

$$x_M = x_A + t(x_B - x_A) \Rightarrow t_M = \frac{x_M - x_A}{x_B - x_A}$$

$$u_M = u_A + \frac{u_B - u_A}{x_B - x_A}(x_M - x_A) \qquad k_3 = \frac{u_B - u_A}{x_B - x_A} \Rightarrow \begin{cases} u_M = u_A + k_3(x_M - x_A) \\ u'_M = u_A + k_3(x_M + 1 - x_A) \end{cases} \Rightarrow \begin{cases} u'_M = u_M + k_3 \\ v'_M = v_M + k'_3 \end{cases}$$

Interpolarea coordonatelor textura (3)

Interpolarea liniara (2)

Generarea unui segment interior triunghiului, de pe linia raster curenta, y=ys

Fie segmentul A(xa,ya,za,ua,va) - B(xb,yb,zb,ub,vb).

```
 dx = 1.0/ (xb - xa);
 dz = (zb - za) * dx; // modificarea coord z la o crestere  a lui x cu 1
 du = (ub - ua) *dx; // k3
 dv = (vb - va) * dx; // k3'
```

Se initializeaza Zbuf[][[] pentru linia curenta la valoarea Zmax (adancimea maxima setata la transformarea in viewport – implicit 1) ;

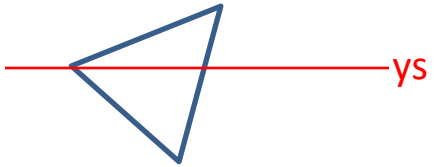
```
pentru (x = xa, z = za, u = ua, v = va; x < xb; x++)
    { //calcul_culoare fragment_in_ fragment_ shader
        *culoare_fragm = combina(culoare_fragm_geom, texture2D(img_textura, u, v));
      daca (z < Zbuf[ys][x]) atunci
       { Zbuf[ys][x] = z;
        * culoare_pixel (x,ys) = culoare_fragm; }
    } u += du; v += dv; z += dz; // u,v,z cresc cu pasii de interpolare liniara
}
```

Interpolarea coordonatelor textura (4)

Interpolarea liniara (3)

Deficienţele interpolarii liniare a coordonatelor textura

1. Pe liniile raster care trec prin varfuri apar discontinuitati ale texturii datorita schimbarii bruste a lui du si dv atunci cand se trece printr-un varf.



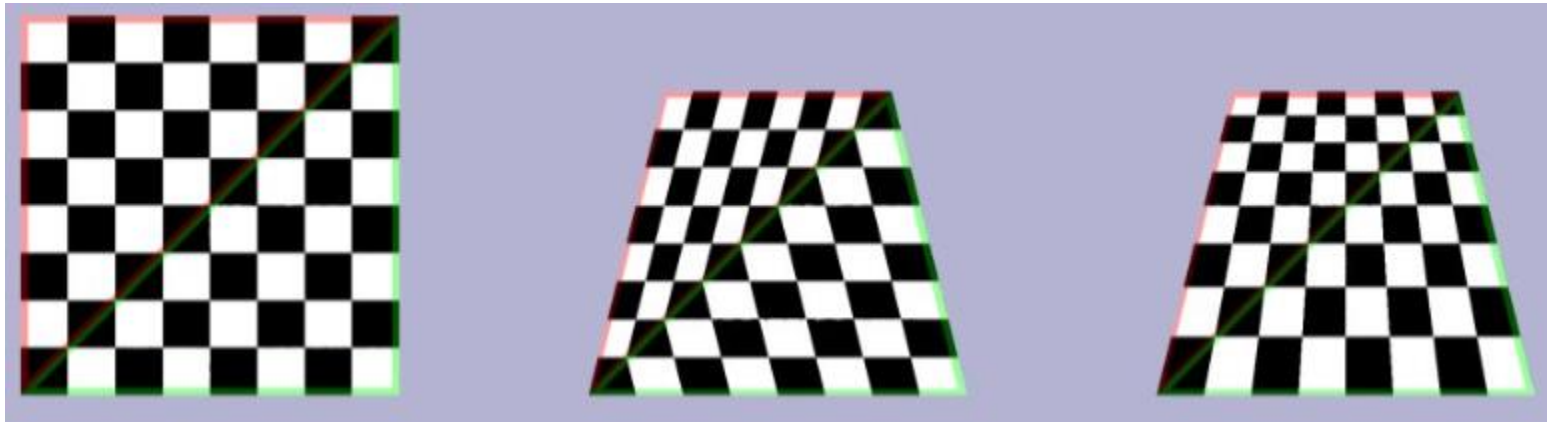
2. Nu produce efectul de scalare specific proiectiei perspectiva

Daca asupra poligonului pe care se aplica textura se efectueaza o proiectie ortografica sau poligonul este paralel cu planul de vizualizare, la incremente egale in spatiul ecran corespund incremente egale in spatiul textura. In consecinta, interpolarea liniara pentru u si v este consistenta cu aceasta transformare spatiala numita <u>mapare afina</u>.

Interpolarea coordonatelor textura (5)

Interpolarea perspectiva (1)

- Atunci cand poligonul nu este paralel cu planul de vizualizare, prin proiectia perspectiva suprafata sa este scalata inegal, proportional cu distanta de la camera (centrul de proiectie), dar nu si imaginea textură.
- <u>Maparea perspectiva</u> produce o scalare inegala a texturii, consistenta cu transformarea perspectiva.

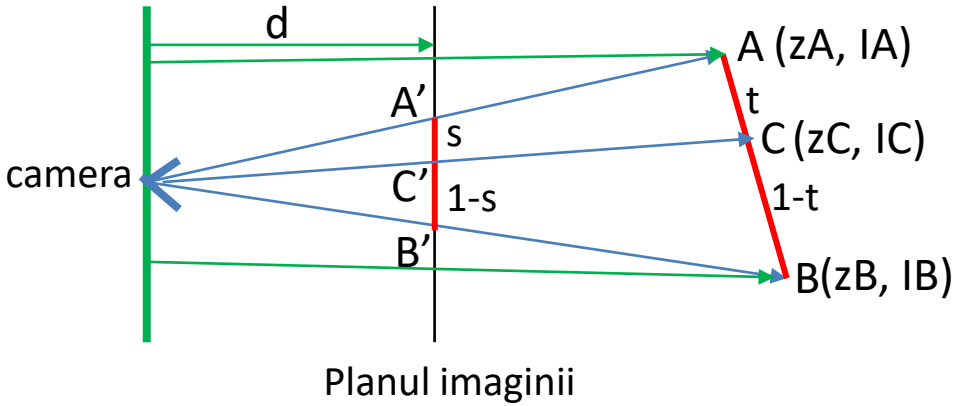


Maparea afină | Maparea afină | Maparea perspectivă

Interpolarea coordonatelor textura (6)

Interpolarea perspectiva (2)

A-B un segm. de dreapta in spatiul camerei
d - distanta la planul de proiectie
I – atributul interpolat, atasat unui varf in spatiul obiect



xA' = -xA/zA
xB' = -xB/zB

Interpolare liniară în spatiul camerei:
In punctul C: IC = IA + t(IB-IA)

Interpolare liniară în spatiul imaginii:

In punctul C': IC' = IA + s(IB – IA)

t/(1-t) ≠ s/(1-s) → IC' nu poate fi obţinut prin interpolare liniara intre IA si IB

Interpolare ne-liniara în spatiul imaginii:

- Se obtine t în functie de s şi distantele zA, zB (din asemanarea triunghiurilor)

$$t = \frac{s*ZA}{s*ZA+(1-s)*ZB} \rightarrow IC' = IA + \frac{s*ZA}{s*ZA+(1-s)*ZB}(IB - IA)$$

Interpolarea coordonatelor textura (7)

Interpolarea perspectiva (3)

- Coordonatele (u,v) nu variaza liniar in spatiul ecran: variatia lor depinde de coordonata z (distanta de la camera: centrul de proiectie)
- 1/z, u/z, v/z variaza liniar (efectul de scalare proportional cu distanta de la camera)
- z este coordonată in spatiul camerei

➢ Interpolarea liniara este convenabila (poate fi efectuata incremental), de aceea:

  - se interpoleaza liniar z'=1/z, u'=u/z, v'=v/z

➢ In maparea afină, u se interpoleaza liniar intre coordonatele extremitatilor, (u0, u1):

$$ut = u0 + t(u1-u0), \quad 0 <= t <= 1$$

➢ In maparea perspectivă:

$$ut = \frac{u0' + t(u1' - u0')}{z0' + t(z1' - z0')} = \frac{ut'}{zt'}$$

unde u0' = u0/z0; u1' = u1/z1; z0' = 1/z0; z1' = 1/z1

➢ Coordonatele z ale extremitatilor, z0, z1, sunt coordonate in spatiul camerei → nu se cunosc la momentul rasterizarii.

Interpolarea coordonatelor textura (8)

Interpolarea perspectiva (4)

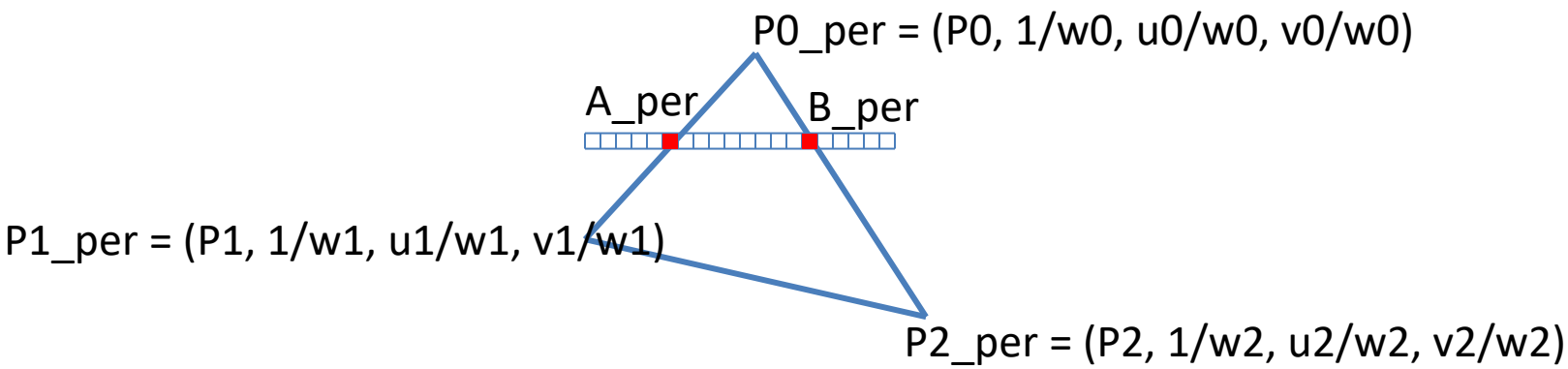
Transformarea de proiectie OpenGL:

$$\begin{bmatrix} xclip \\ yclip \\ zclip \\ wclip \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} xcamera \\ ycamera \\ zcamera \\ 1 \end{bmatrix}$$

wclip = - zcamera → coordonata z a punctului proiectat

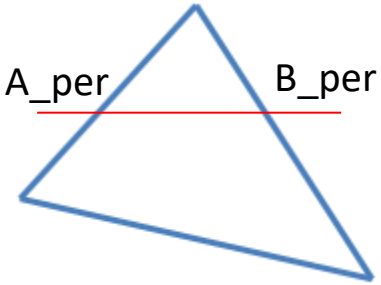
z'=1/z, u'=u/z, v'=v/z → z = wclip : distanta punctului proiectat fata de camera

Deci, în interpolare se poate folosi coord. wclip a punctului proiectat în loc de coord z.



Interpolarea coordonatelor textura (9)

Interpolarea perspectiva (5)



Interpolarea perspectiva efectuata de GPU.

Fie segmentul A_per - B_per:

A_per(xa,ya,za,wa_per=1/wa, ua_per=ua/wa, va_per =va/wa)

B_per(xb,yb,zb,wb_per=1/wb, ub_per=ub/wb, vb_per =vb/wb) .

Pentru fiecare pixel (xp, yp) de pe segmentul A_per – B_per:

- se obțin wp', up', vp' prin interpolare liniară între 1/wA – 1/wB, uA/wA – uB/wB, vA/wA – vB/wB
- se calculeaza coordonatele (up,vp) corectate:
  - ucorect = u'p/w'p = (up/wp)/(1/wp) = up, vcorect = v'p/w'p = (vp/wp)/(1/wp) = vp.

➢ Prin aceasta corectie, în acele parti ale poligonului mai apropriate de observator, diferenta între coordonatele textura de la pixel la pixel este mai mica (întinzand textura) iar în partile mai îndepartate diferenta este mai mare, comprimand textura.

Interpolarea coordonatelor textura (10)

Interpolarea perspectiva (6)

Fie segmentul A_per(xa,ya,za,wa_per, ua_per, va_per) - B_per(xb,yb,zb,wb_per, ub_per, vb_per)

Se initializeaza Zbuf[][[] pentru linia curenta la valoarea Zmax a volumului de vizualizare;

```
dx = 1.0/ (xb - xa); dz = (zb - za) * dx; dw = (wb_per - wa_per)*dx;

du = (ub_per – ua_per) *dx; dv = (vb_per – va_per) * dx;

for (x=xa, u=ua_per, v=va_per, w=wa_per; x<xb; x++)
   { //calcul_culoare fragment_in_ fragment_ shader
         *culoare_fragm = combina(culoare_fragm_geom, texture2D(img_textura, u/w, v/w));

     daca (z<ZBuf[ys][x]) atunci

      { ZBuf[ys][x]=z;   *culoare_pixel (x,ys) = culoare_fragm; }

        z+=dz; u+=du; v+=dv; w+=dw;

   }
```

Interpolarea coordonatelor textura (11)

Interpolarea perspectiva (7)

CONCLUZIE: Interpolarea perspectiva

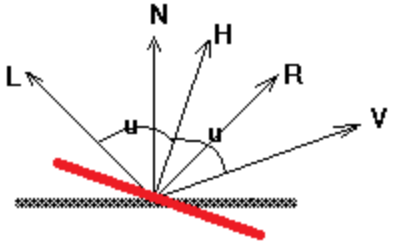
- Este necesara numai pentru proiectia perspectiva; pentru proiectia paralela interp. liniara in spatiul ecran corespunde interp. liniare in spatiul camerei.
- Nu este necesara pentru linii si poligoane paralele cu planul de proiectie (planul “near”).
- Poate fi aplicata tuturor atributelor atasate unui varf: culoare, normala, coord. textura.
- wclip poate fi pastrata la transformarea din coord. de decupare in coord disp. normalizate si atasata varfului.
- Modul implicit de interpoare pe placile grafice moderne (cu versiunile actuale OpenGL) este cel perspectiva.
- Se poate specifica modul de interpolare prin: glHint(GL_PERSPECTIVE_CORRECTION_HINT, hint);
  unde hint poate fi: GL_NICEST: Perspectiva; GL_FASTEST: Lineara, GL_DONT_CARE: indiferent
  Daca interpolarea perspectiva nu este suportata eficient de placa grafică, GL_DONT_CARE sau GL_FASTEST au rezultat interpolarea liniara a culorilor si coordonatelor textura.

Redarea suprafetelor 3D folosind texturi- 3

Prof. univ. dr. ing. Florica Moldoveanu

Curs *Sisteme de Prelucrare Grafică* – UPB, Automatică şi Calculatoare
2021-2022

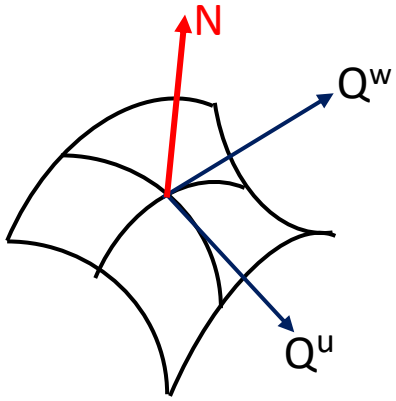
Simularea detaliilor fizice ale suprafetelor ("Bump Mapping")



- Calculul reflexiei luminii este bazat pe normala la suprafata.
- Normalele in puncte vecine ale unei suprafete netede difera putin.
- Modificand normalele se pot obtine variatii mari ale intensitatii luminii reflectate in puncte vecine, care ar corespunde unor variatii mari ale geometriei suprafetei (exemplu, sfera din stanga)

Bump mapping(2)

- Metoda introdusa de J. Blinn in 1978.
- Modificarea normalei intr-un punct al unei suprafete este definita printr-o “functie de perturbare”.
- Fie Q(u,w) ecuatia parametrica a suprafetei.



Q^u si Q^w sunt vectorii tangenti la suprafata in punctul Q(u,w)

$$Q^u = \frac{\partial Q(u,w)}{\partial u}$$

$$Q^w = \frac{\partial Q(u,w)}{\partial w}$$

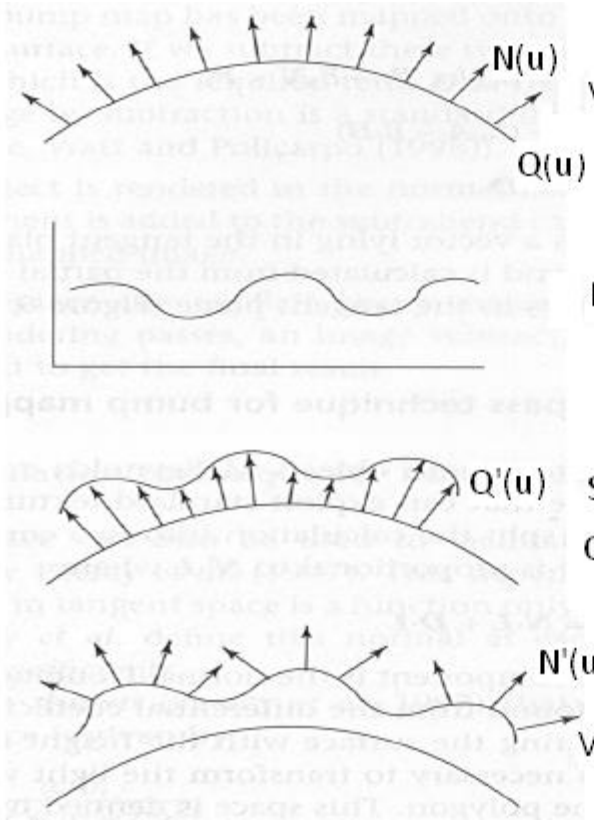
$$\overline{N} = \overline{Q}^u \times \overline{Q}^w$$

Suprafata perturbata: $Q'(u,w) = Q(u,w) + P(u,w)\,\frac{\overline{N}(u,w)}{|N(\overline{u},w)|}$

***P(u,w)* este functia de perturbare**

Bump mapping(3)

Interpretarea geometrica 1D a metodei



Vectorii normali la suprafata originala

Functia de perturbare: P(u)

Suprafata perturbata:

Q'(u) = Q(u) + P(u)*N(u)/|N(u)|

Vectorii normali la suprafata perturbata

Bump mapping(4)

- Normala la suprafata perturbata, intr-un punct (u,w), N'(u,w), se obtine ca produs vectorial al vectorilor tangenti la suprafata perturbata:

$$\begin{cases} Q'^{u} = Q^{u} + P^{u}\dfrac{N}{|N|} + P\dfrac{N^{u}}{|N|} \\ Q'^{w} = Q^{w} + P^{w}\dfrac{N}{|N|} + P\dfrac{N^{w}}{|N|} \end{cases}$$

$$N' = Q'^{u} \times Q'^{w}$$

- P(u,v) are valori mici, de aceea ultimul termen se poate neglija.

  Rezulta:

$$\overline{N}' = \overline{Q}'^{u} \times \overline{Q}'^{w} = Q^{u} \times Q^{w} + \frac{P^{u}\left(\overline{N} \times \overline{Q}^{u}\right)}{|N|} + \frac{P^{w}\left(\overline{Q}^{w} \times \overline{N}\right)}{|N|} + \frac{P^{u}P^{w}\left(\overline{N} \times \overline{N}\right)}{|N|^{2}} \Rightarrow \overline{N}' = \overline{N} + \overline{D}$$

- Se poate folosi ca functie de perturbare orice functie derivabila.
- Functia de perturbare poate fi aproximata printr-un tablou bi-dimensional adresat prin valori discrete (u,v); valorile functiei pentru valori intermediare ale lui u sau v se obtin prin interpolare bi-liniara iar derivatele partiale se aproximeaza prin diferente finite.

Bump mapping(5)

- Efectul texturii este invariant fata de translatie dar nu si fata de scalarea obiectului pe care se aplica: deoarece textura nu se modifica la scalarea suprafetei, apare efectul de aplatisare a texturii.

  $N' = N + D \rightarrow$ D depinde de N, Q^u , Q^w

- Pentru ameliorarea efectului de aplatisare , Blinn a sugerat urmatoarea corectie:

$$\overline{N'} = \overline{N} + \overline{D'} \qquad \overline{D'} = a \cdot \overline{D} \cdot \frac{|N|}{|D|} \qquad a = \sqrt{P^{u^2} + P^{w^2}}$$

Implementarea moderna a metodei Bump mapping foloseste hărți de normale

https://learnopengl.com/Advanced-Lighting/Normal-Mapping

Harti de normale (1)

❖ O harta de normale (normal map) este o imagine textura ale carei elemente sunt vectori 3D, folositi ca normale in calculul culorii fragmentelor.

❖ Hartile de normale se pot obtine in 2 moduri:

1) Se calculeaza normalele la modelul detaliat si se memoreaza intr-o textura, apoi se simplifica geometria modelului.

- Se poate folosi pentru aceasta aplicatia nVidia Melody: http://www.nvidia.com/object/melody_home.html.
- Majoritatea aplicatiilor de modelare (3ds Max, Maya, Blender, s.a) pot produce harti de normale

2) Folosind hărți de înaltimi

Harti de normale (2)

Codificarea normalelor în textura

- Componentele unui vector normal normalizat: -1<= nx,ny,nz <=1
- Componentele culorii dintr-un texel RGB: 0<= R,G,B <= 1
- Codificarea normalelor in textura: RGB (nx,ny,nz) = (nx,ny,nz) / 2 + (0.5,0.5,0.5)
- Conversia culoare texel → vector normal: N(R,G,B) = 2*(R,G,B) – (1,1,1)
  - Exemple: N(0.5,0.5,1) = (0,0,1) – vectorul normal la planul texturii

    N(0.43, 0.91, 0.80) = (–0.14, 0.82, 0.6)

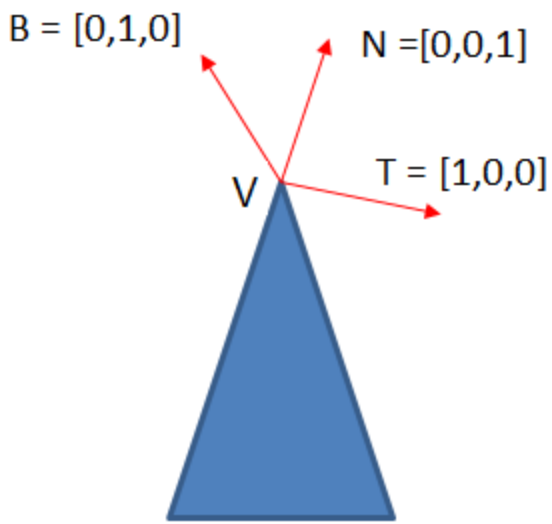
❖ Normalele dintr-o textura "harta de normale" sunt definite într-un spatiu (sistem de coordonate) diferit de spatiul obiect al obiectului pentru care se foloseste harta, a.î. normalele sa nu fie afectate de rotatia aplicata obiectului. Daca normalele sunt relative la fiecare față a obiectului, nu sunt afectate de rotatiile aplicate obiectului.

❖ Normalele dintr-o textura sunt raportate la "**spațiul tangent**", definit pentru fiecare față a mesh-ului pe care se aplica.

Harti de normale (3)

Spațiul tangent

- Spatiul tangent al unei fețe este definit printr-un sistem de coordonate 3D, determinat de 3 vectori ortogonali:
  - Vectorul normal atasat unui vârf al feței, N, care da directia axei OZ a spatiului tangent
  - Un vector tangent la suprafață în vârf, T, care da directia axei OX a spatiului tangent
  - Vectorul obtinut ca produs vectorial între N şi T, numit vectorul bitangent, B



- Fie T,B,N directiile axelor sistemului de coordonate al spatiului tangent in varful V
- N are directia normalei geometrice in varf
- T si B sunt vectori tangenti la suprafata din care face parte triunghiul, in varful V

Harti de normale (4)

- N puncteaza in sus (in afara planului texturii): reprezinta axa z.
- Vectorul T (Tangent) se alege a.i. sa fie tangent la suprafață în punctul considerat si sa aiba directia pozitiva a axei u a texturii.
- Vectorul B (Bitangent) trebuie sa fie tangent la suprafata si impreuna cu T si N sa formeze un sistem de coord carteziene dreapta; B are directia pozitiva a axei v a texturii.
- Vectorii T si B se obtin din urmatoarele ecuatii, in care V0, V1, V2 sunt vectorii coordonatelor celor 3 varfuri ale unui triunghi, iar (ui,vi) coordonatele textura atasate varfurilor:

$$V1 - V0 = (u1 - u0) * T + (v1-v0) * B$$

$$V2 - V0 = (u2 - u0) * T + (v2-v0) * B$$

- Vectorii N, T si B se asociaza fiecarui varf al mesh-ului care aproximeaza suprafata texturata. Ei sunt raportati la spatiul obiect (in care sunt reprezentate coordonatele varfurilor).
- Prin aplicarea transformarii de modelare, (T,B,N) sunt transformati in sistemul coordonatelor globale (spatiul scenei 3D).

Harti de normale (5)

❖ Pentru calculul luminii reflectate (in fragment shader) este necesar ca vectorii V (directia observatorului), L (directia sursei de lumina) si N sa fie exprimati in acelasi sistem de coordonate. L si V sunt definiti in spatiul scenei. Exista 2 posibilitati:

1. Se transforma vectorul normal extras din textura din spatiul tangent in spatiul scenei
2. Se transforma vectorii L si V din spatiul scenei in spatiul tangent.

- Pentru conversia unui vector din spatiul tangent în spatiul scenei se poate folosi matricea TBN, in care vectorii T,B,N sunt raportati la spatiul scenei:

$$TBN = \begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix}$$

- L si V se transforma din spatiul scenei in spatiul tangent prin transformarea inversa - matricea inversa a matricei TBN, care este aceeasi cu transpusa matricei TBN;

  invTBN = transpTBN

- Vectorii L si V pot fi transformati fie in vertex shader fie in fragment shader (prima varianta este avantajoasa d.p.d.v al calculelor).

Harti de normale (6)

Vertex shader

```
layout (location = 0) in vec3 vPos;
layout (location = 2) in vec3 vNormal;
layout (location = 3) in vec2 vTexCoords;
layout (location = 4) in vec3 vTangent;
layout (location = 5) in vec3 vBitangent;
out VS_OUT {
   vec2 TexCoords;
   vec3 TangentLightPos;//pozitia sursei in spatiul tangent al varfului
   vec3 TangentViewPos; //pozitia observatorului in spatiul tangent al varfului
   vec3 TangentFragPos; //coord fragment in spatiul tangent
} vs_out;
uniform mat4 projection;uniform mat4 view;uniform mat4 model;
uniform vec3 lightPos; uniform vec3 viewPos;
void main()
{   gl_Position = projection * view * model * vec4(vPos, 1.0);
   vs_out.TexCoords = vTexCoords;
   vec3 T   = normalize(mat3(model) * vTangent);
   vec3 B   = normalize(mat3(model) * vBitangent);
   vec3 N   = normalize(mat3(model) * vNormal);
   mat3 TBN = transpose(mat3(T, B, N));//spatiul scenei → spatiul tangent
   vs_out.TangentLightPos = TBN * lightPos;
   vs_out.TangentViewPos  = TBN * viewPos;
   vs_out.TangentFragPos  = TBN * vec3(model * vec4(vPos, 1.0));
}
```

```
struct VertexFormat {
glm::vec3 Position;
glm::vec3 color;
glm::vec3 Normal;
glm::vec2 TexCoords;
glm::vec3 Tangent;
glm::vec3 Bitangent;
};
```

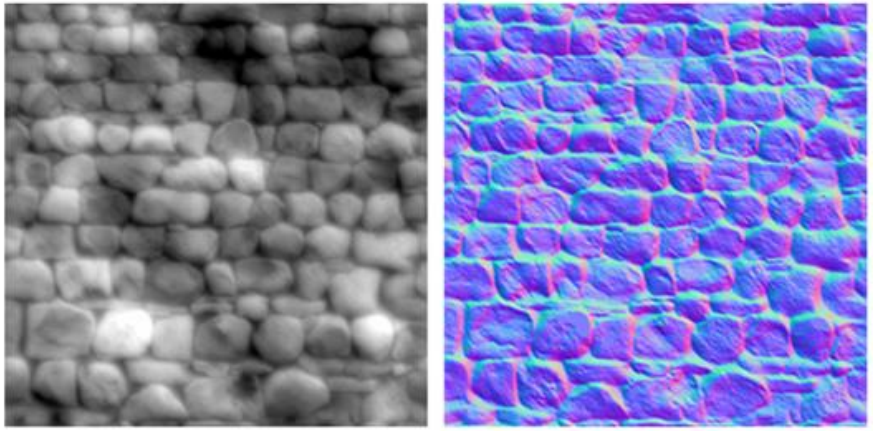
Harti de normale (6)

Fragment shader

```
in VS_OUT {
   vec2 TexCoords;
   vec3 TangentLightPos;//Poz sursa in spatiul tangent
   vec3 TangentViewPos; //Poz observator in spatiul tangent
   vec3 TangentFragPos;//Poz fragment in spatiul tangent
} fs_in;
layout(location = 0) out vec4 out_color;
void main()
{vec3 normal = texture(normalMap, fs_in.TexCoords).rgb;
 normal = normalize(normal * 2.0 - 1.0);// transf culoarea in vector si se normalizeaza
 vec3 lightDir = normalize(fs_in.TangentLightPos - fs_in.TangentFragPos);//directie sursa lumina
 vec3 viewDir =  normalize(fs_in.TangentViewPos-fs_in.TangentFragPos); //directie obs.
*calculeaza culoarea fragmentului folosind vectorii normal, lightDir, viewDir
  out_color = culoare;
}
```

Harti de inaltimi (1)

- Intr-o imagine **harta de inaltimi (hight map)**, fiecare pixel reprezinta inaltimea unui punct de pe o suprafata.
- Hartile de înaltimi se folosesc in mai multe metode de texturare pentru simularea iregularitatilor unei suprafete, pentru generarea terenurilor, dar si in alte aplicatii (GIS)
- Inaltimile sunt codificate prin niveluri de gri: 0 - negru, 1 - inaltimea maxima - alb .
- In implementarea metodei Bump mapping, hartile de inaltimi se folosesc pt generarea hartilor de normale.



Harta de inaltimi Harta de normale

- Calculul normalelor din harta de inaltimi se poate face la momentul calculului culorii fragmentelor (in fragment shader) sau intr-o etapa de preprocesare.

Harti de inaltimi (2)

Calculul normalelor din harta de inaltimi

- Vectorul normal intr-un punct al unei suprafete este definit ca produsul vectorial al celor doi vectori tangenti la suprafata in acel punct.
- Fie HI harta de inaltimi. Cei doi vectori tangenti intr-un punct (x,y) se aproximeaza prin diferente finite astfel:

  H1= (1, 0, Hx), Hx = HI[x+1, y] - HI[x, y] – variatia inaltimii pe directia x

  H2 = (0, 1, Hy), Hy = HI[x, y+1] - HI[x, y] – variatia inaltimii pe directia y

- Normala normalizata in punctul (x,y) este :

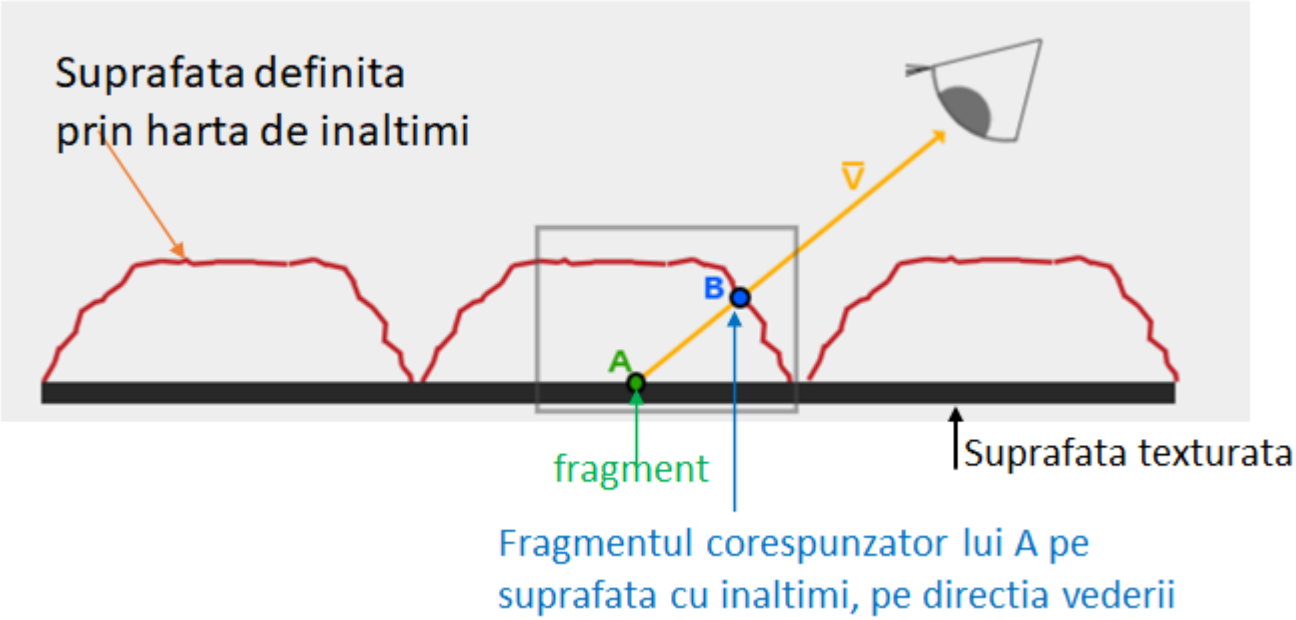
  $Nu = H1xH2/|H1xH2| = (-Hx, -Hy, 1)/sqrt(Hx^2 + Hy^2 +1)$

Parallax mapping(1)

- Este o imbunatatire a metodei Normal Mapping prin marirea efectului de adancime pe suprafete cum ar fi pereti din caramida sau piatra.
- Metoda nu afecteaza performanta (in plus față de Normal Mapping), de aceea se foloseste in jocurile video.
- A fost introdusa de Tomomichi Kaneko et al., in 2001 (Kaneko, T., et al., 2001. Detailed Shape Representation with Parallax Mapping. Proceedings of ICAT 2001, pp. 205-208.)

❖ Metoda foloseste o textura harta de înalțimi, dar nu modifica coordonatele fragmentelor suprafetei pe care se aplica textura, ci creaza iluzia ca fragmentele se afla la diferite inaltimi alterand coordonatele textura ale fragmentelor in functie de directia vederii in fiecare fragment si înaltimea din harta de înaltimi pentru acel fragment.

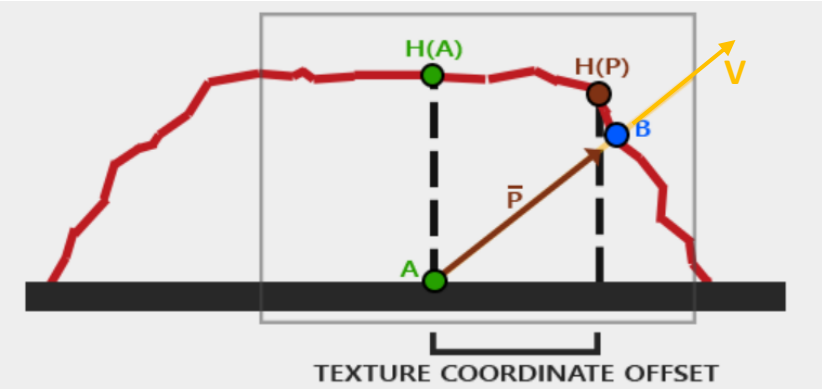
Parallax mapping(2)

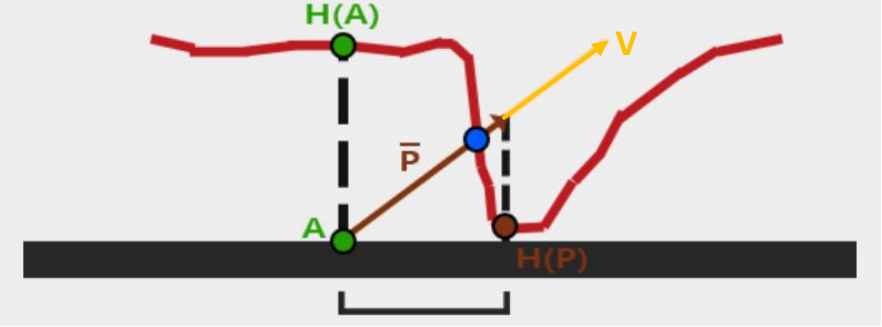


- Deoarece suprafata texturata nu este deformata pe baza hartii de inaltimi, vectorul directie de proiectie (V) este calculat din punctul A.
- Metoda Parallax simuleaza deplasarea punctului A in B modificand coord textura ale fragmentului A astfel incat sa obtinem coordonatele textura ale lui B.

Parallax mapping(3)

- Folosind in A coord. textura ale lui B se creaza efectul ca observatorul priveste catre B.
- Pentru aceasta maparea parallax scaleaza lungimea vectorului V a.i. sa fie egala cu H(A).
- Vectorul scalat este P [dx_P, dy_P, dz_P), lung(AP) = H(A).
- dx_P, dy_P se folosesc ca offset-uri pt coord textura ale fragmentului A. Cu cat H(A) este mai mare cu atat mai mult este “deplasat” fragmentul A.
- Punctul de pe suprafata cu inaltimi (B) este aproximat prin varful vectorului P. Aceasta aproximare da rezultate bune in cele mai multe cazuri dar atunci cand inaltimile din H au variatii mari, rezultatele obtinute nu sunt realiste, pt ca P nu este suficient de aproape de B.





Parallax mapping(4)

- Pentru ca textura sa fie independenta de rotatiile obiectului pe care se aplica, vectorul P trebuie sa fie raportat la un sistem de coordonate diferit de cel al obiectului, ca si in cazul vectorilor dintr-o harta de normale. → Maparea parallax se face in spatiul tangent.

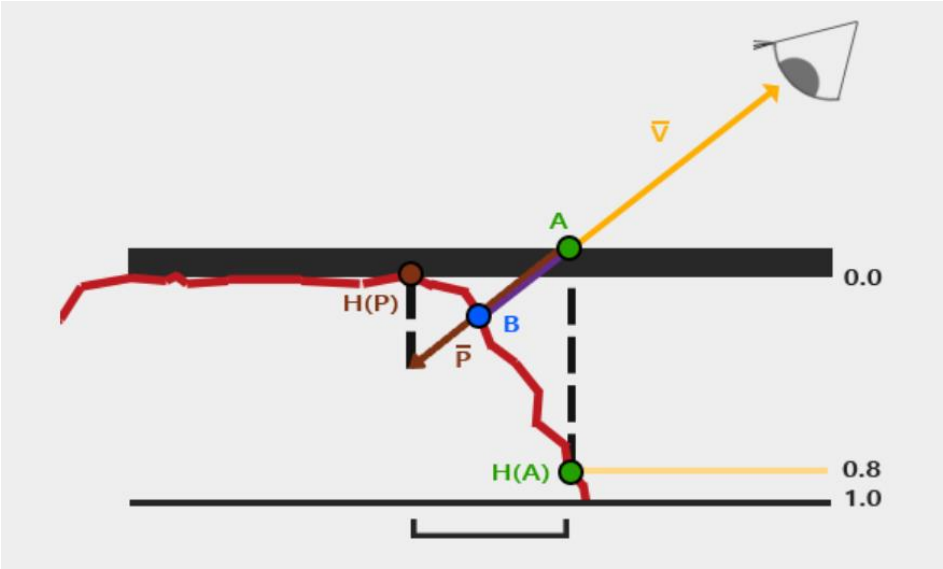
- Transformand vectorul V in spatiul tangent, vectorul P va avea componentele dx_P, dy_P aliniate cu vectorii tangent (T) si bitangent (B) ai feței pe care se aplica textura.
- Doarece T si B au directia coordonatelor u si v, se pot folosi componentele dx_P, dy_P ca offset-uri pt coordonatele textura, indiferent de orientarea suprafetei.

- Scopul maparii Parallax este de a crea iluzia unei suprafete cu variatii ale inaltimii: iluminarea trebuie sa ajute in crearea acestei iluzii.

- Folosirea unei hărți de normale obtinuta din harta de inaltimi asigura o iluminare coerenta cu variatia inaltimilor.

Parallax mapping(5)

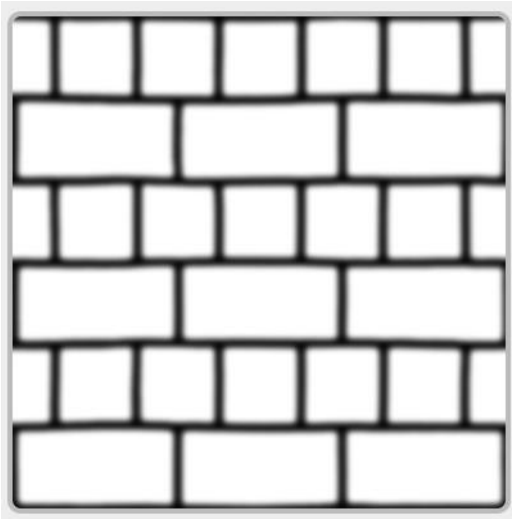
- In parallax mapping se prefera sa se foloseasca inversa hartii de inaltimi, fiind mai usor de creat iluzia de denivelari pe adancime decat pe inaltime pe o suprafata plata. In acest caz perceperea maparii parallax este:



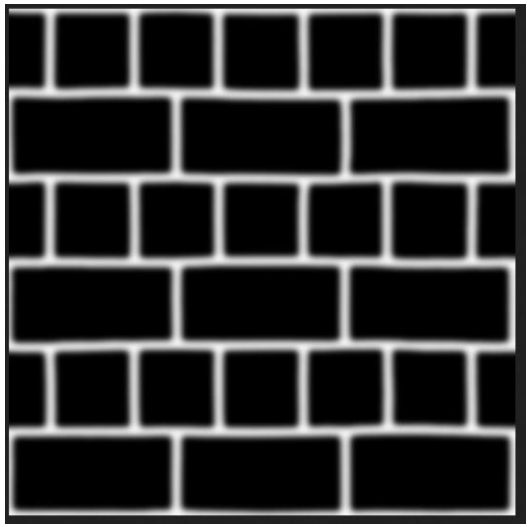
- In acest caz cordonatele textura ale punctului A se modifica scazand din ele componentele vectorului P.

Parallax mapping(6)

- Valoarea de adancime pt fragmentul A = 1- H(A) – poate fi calculata in fragment shader sau se poate crea inversa hartii de adancime intr-un soft de editare.



Harta de inaltimi



Inversa hartii de inaltimi

- Maparea parallax este implementata in fragment shader deoarece efectul de denivelare difera pe suprafata unei fete (triunghi) de la fragment la fragment.

Parallax mapping(7)

- In fragment shader trebuie sa se calculeze vectorul V, deci avem nevoie de pozitia observator (View) si pozitia fragmentului in spatiul tangent.
- Se poate folosi programul vertex shader din implementarea “normal mapping”.
- Functia ParallaxMapping din fragment shader implementeaza modificarea coord textura ale fragmentului.
- Are ca parametrii coord textura ale fragmentului si vectorul V in spatiul tangent.
- Functia intoarce coordonatele textura modificate, care sunt apoi utilizate pentru extragerea culorii din textura difuza si vectorul normal din harta de normale.

➢ Culoarea difuza si vectorul normal corespund suprafetei cu geometria modificata prin H .

```
vec2 ParallaxMapping(vec2 texCoords, vec3 viewDir)
{
   float height =  texture(depthMap, texCoords).r; // extrage adancimea fragmentului din
                                                   // depthMap
   vec2 p = viewDir.xy / viewDir.z * (height * height_scale);// P= TangentViewDir*H(frgm)
   return texCoords - p;    //folosind inversa hartii de inaltimi
}
```

Parallax mapping(8)

Fragment shader

```
out vec4 FragColor;
in VS_OUT {
    vec2 TexCoords;
    vec3 TangentLightPos;
    vec3 TangentViewPos;
    vec3 TangentFragPos;
} fs_in;

uniform sampler2D diffuseMap;//textura difuza
uniform sampler2D normalMap;
uniform sampler2D depthMap;

uniform float height_scale;

vec2 ParallaxMapping(vec2 texCoords, vec3 viewDir);

void main()
{
    vec3 viewDir   = normalize(fs_in.TangentViewPos - fs_in.TangentFragPos);
    vec2 texCoords = ParallaxMapping(fs_in.TexCoords, viewDir);
    // extrage culoarea si normala fragm. folosind cord. textura mdificate
    vec3 diffuse = texture(diffuseMap, texCoords);
    vec3 normal  = texture(normalMap, texCoords);
    normal = normalize(normal * 2.0 - 1.0);//transforma (R,G,B)in vector
    // se calclueaza FragColor folosind vectorii normal, viewDir, lightDir
    ....
}
```

Parallax mapping(9)

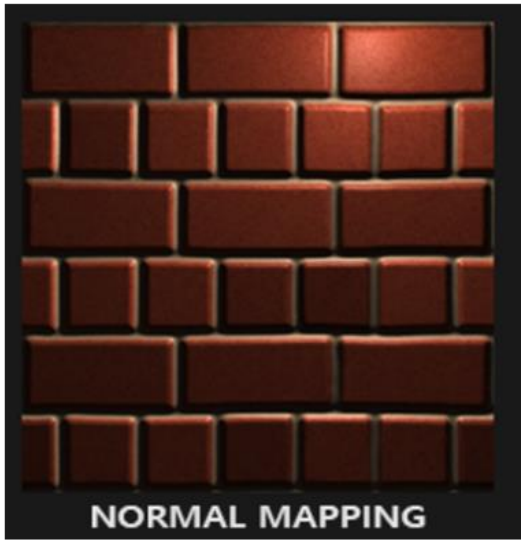
- In functia ParallaxMapping se calculeaza vectorul P scaland vectorul View din spatiul tangent cu adancimea fragmentului preluata din harta depthMap.

```
vec2 p = viewDir.xy / viewDir.z * (height * height_scale);
return texCoords – p;
```

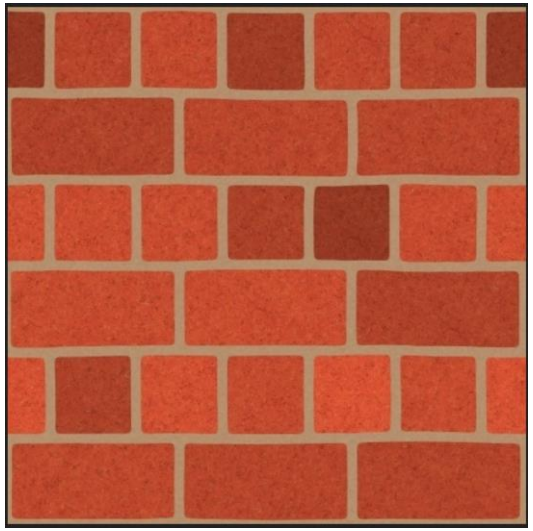
- viewDir.z are valori [0,1]
- Atunci cand viewDir este aproape paralel cu suprafata texturata, componenta z este aproape de zero (viewDir este in spatial tangent) si vectorul P va fi mult mai mare decat atunci cand viewDir este aproape perpendicular pe suprafata.
- Prin factorul de scalare height_scale se ajusteaza marimea lui P, astfel incat sa modifice mai mult coordonatele textura atunci cand suprafata este privita cu un anumit unghi decat atunci cand este privita aproape de sus: dă rezultate mai realiste.

- In varianta *Parallax Mapping with Offset Limiting* este eliminata impartirea prin viewDir.z, pentru evitarea efectelor nadorite la anumite unghiuri.

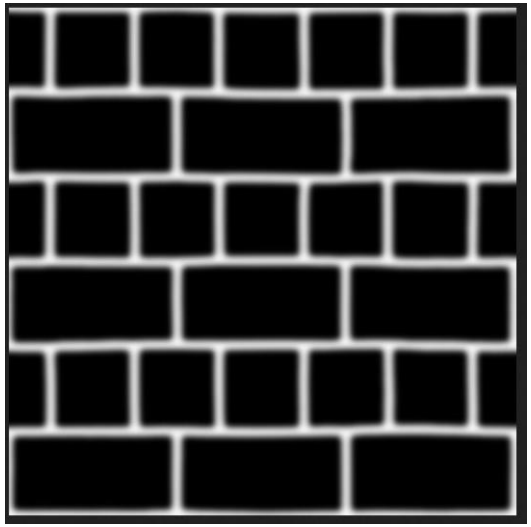
Parallax mapping(10)



https://learnopengl.com/Advanced-Lighting/Parallax-Mapping



diffuseMap



depthMap

Parallax mapping(11)

- La marginile suprafetei texturate prin maparea Parallax pot aparea artefacte atunci cand coordonatele textura modificate ies din domeniul [0,1].
- Aceste cazuri se pot evita prin eliminarea fragmentului in fragment shader:

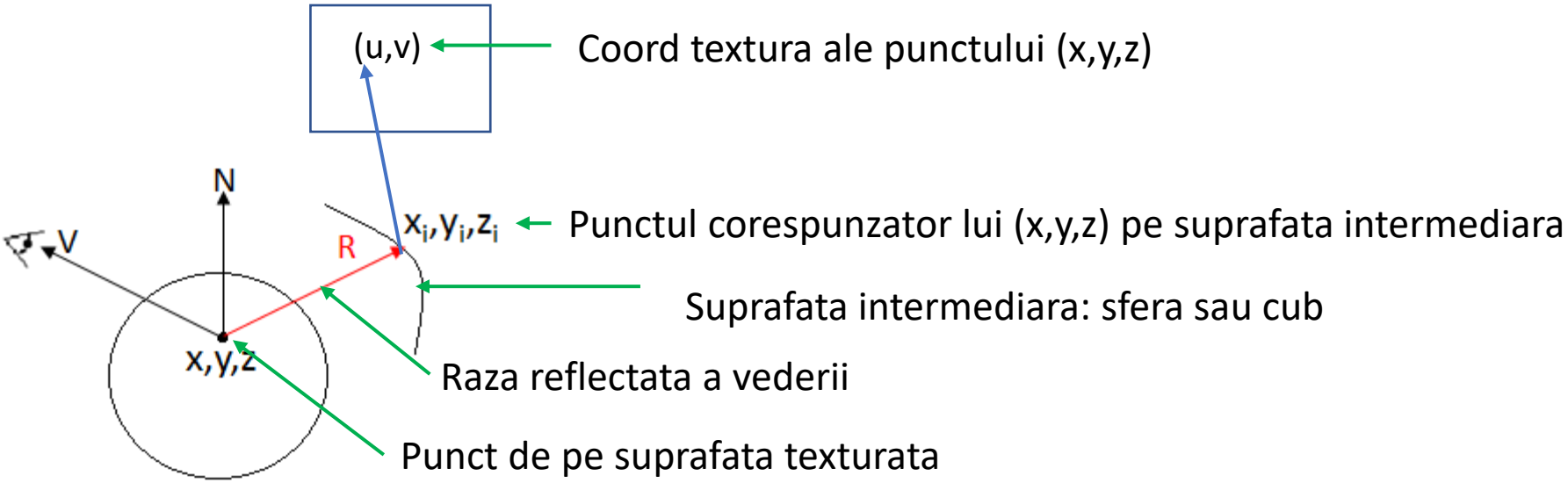
```
texCoords = ParallaxMapping(fs_in.TexCoords,  viewDir);
if(texCoords.x > 1.0 || texCoords.y > 1.0 || texCoords.x < 0.0 || texCoords.y < 0.0)
   discard;
```

Maparea hartii mediului inconjurator ("Environment mapping")

- Este o tehnica de mapare care tine cont de pozitia observatorului.
- Ideea: de a simula ceea ce vede un observator pe suprafetele obiectelor din scena 3D daca acestea au suprafete care reflecta/refracta lumina, atunci cand se deplaseaza in scena 3D.
- Se foloseste ca imagine textură imaginea mediului inconjurator obiectului/obiectelor pe care este redata textura.
- Imaginea textura este aplicata pe suprafetele obiectelor scenei printr-o "mapare in 2 pasi": ca suprafata intermediara se foloseste o sfera sau un cub.
- Daca obiectul texturat este reflectant (suprafata sa reflecta specular lumina incidenta) atunci se simuleaza reflexia hartii textura de catre suprafata obiectului.
- Daca obiectul texturat este transparent, atunci harta textura este redata pe suprafata obiectului prin simularea fenomenului de refractie.

Environment mapping(2)

Mapare in 2 pasi dependenta de pozitia observatorului



- Imaginea redata pe suprafata obiectului texturat este dependenta de pozitia observatorului, căci raza reflectata/refractata este calculata in functie de pozitia observatorului si normala la suprafata.

Environment mapping(3)

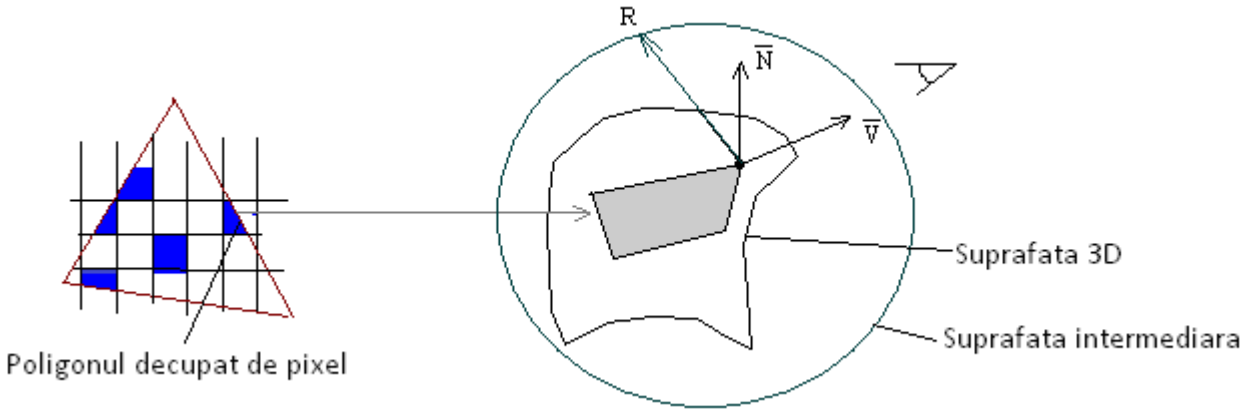
Metoda exacta(1)

Culoarea unui fragment se obtine astfel:

1. Se transforma varfurile fragmentului din spatiul ecran in sistemul de coordonate globale, rezultand un poligon (poate fi o suprafata curba) pe suprafata obiectului texturat.

2. Se calculeaza normalele in varfurile poligonului de pe suprafata texturata.

3. Se construieste "spotul reflectat (sau refractat) al vederii", alcatuit din vectorii reflectati (refractati) ai vederii, R_i, în varfurile poligonului. Acestia se calculeaza folosind normala (N_i) in fiecare varf al poligonului si pozitia observatorului (V):

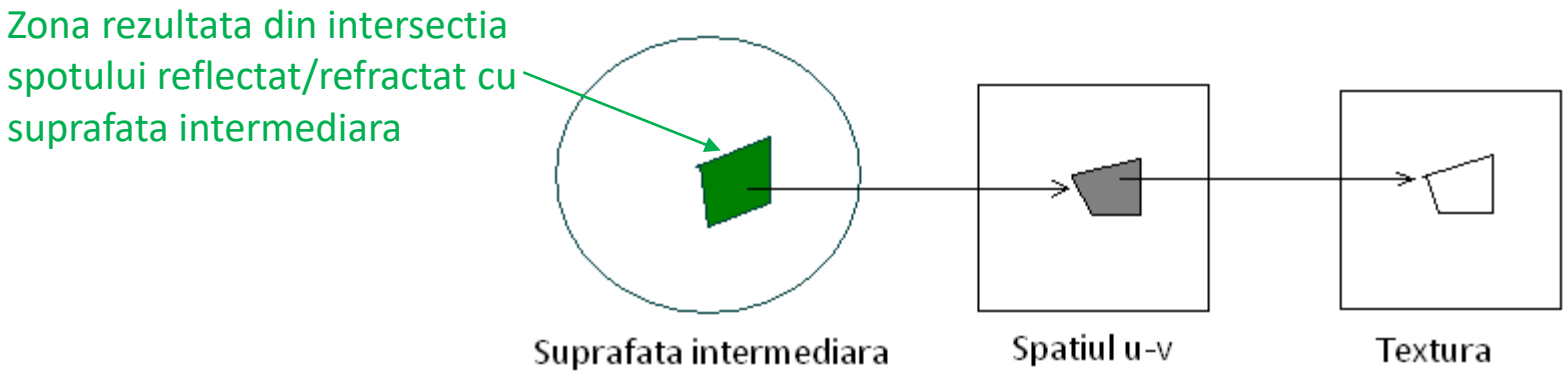


Environment mapping(4)

Metoda exacta(2)

4. Se intersecteaza suprafata intermediara cu spotul reflectat (sau refractat). Prin maparea inversa a suprafetei de intersectie se determina zona u-v apoi zona din spatiul textura, din care se calculeaza culoarea ce va fi folosita la afisarea fragmentului.

5. Culoarea rezulta prin compresia texturii din zona.



Environment mapping(5)

Metoda practica

Se calculeaza o singura raza reflectata/ refractata pentru fiecare fragment.

1. Normala fragmentului (obtinuta prin interpolare in procesul de rasterizare) trebuie sa fie raportata la sistemul coordonatelor globale.
2. Se calculeaza raza reflectata/ refractata folosind normala fragmentului si pozitia observatorului (tot in coord. globale)
3. Se calculeaza intersectia razei cu suprafata intermediara pe care se aplica harta mediului inconjurator (cubul sau sfera).
4. Se calculeaza coordonatele textura care determina culoarea fragmentului, folosind coordonatele punctului de intersectie si functia de mapare inversa (sferica sau plana).

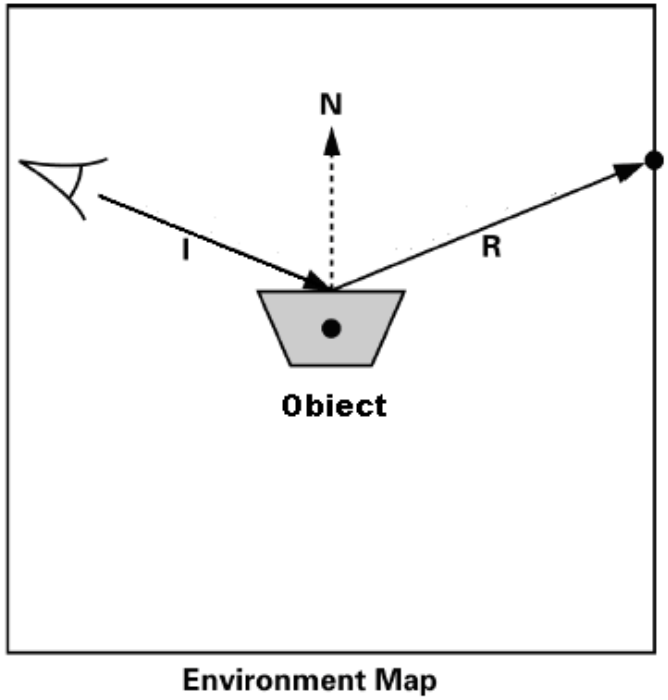
Metoda presupune ca mediul inconjurator al unui obiect se afla la distanta infinita de el si deci poate fi reprezentat ca o imagine omni-directionala. Aceasta deoarece harta mediului este accesata printr-un singur vector 3D.

Environment Mapping(6)

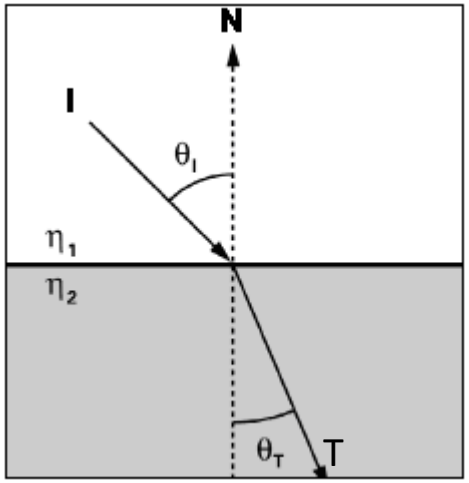
Calculul razei reflectate/rafractate pentru un fragment

- Se folosesc: normala fragmentului si pozitia observatorului, în coordonate globale.
- Vectorul reflectat (R) sau cel transmis (T) este intersectat cu suprafata intermediara.
- Deoarece conteaza numai directia vectorului reflectat/refractat si nu mărimea lui, suprafata intermediara este cubul/ sfera unitate, centrat/a in originea sist de coord globale iar vectorul R/T pleaca din origine.

Reflexia



Refractia



Directia razei transmise se calculeaza pe baza legii lui Snell; n1, n2 – indicii de refractie ai celor 2 medii.

Environment mapping(7)

Limitarile metodei

- In mod ideal, fiecare obiect dintr-o scena trebuie sa aiba propria harta textura a mediului inconjurator. In practica, mai multe obiecte pot partaja aceeasi harta a mediului inconjurator, fara ca acest aspect sa se observe.
- Harta textura ar trebui sa fie modificata atunci cand obiectele scenei isi schimba pozitia sau cand un obiect isi schimba semnificativ pozitia relativ la mediu. In practica, se obtin reflexii foarte bune cu harti ale mediului statice.
- Obiectele pot reflecta numai mediul codificat prin harta textura. Nu pot reflecta parti proprii. De aceea, metoda este adecvata pentru obiecte convexe sau aproape convexe.
- Nu pot fi redate reflexii multiple: doua obiecte stralucitoare care se reflecta unul pe celalalt.

Environment mapping(8)

Cube Environment Mapping(1)

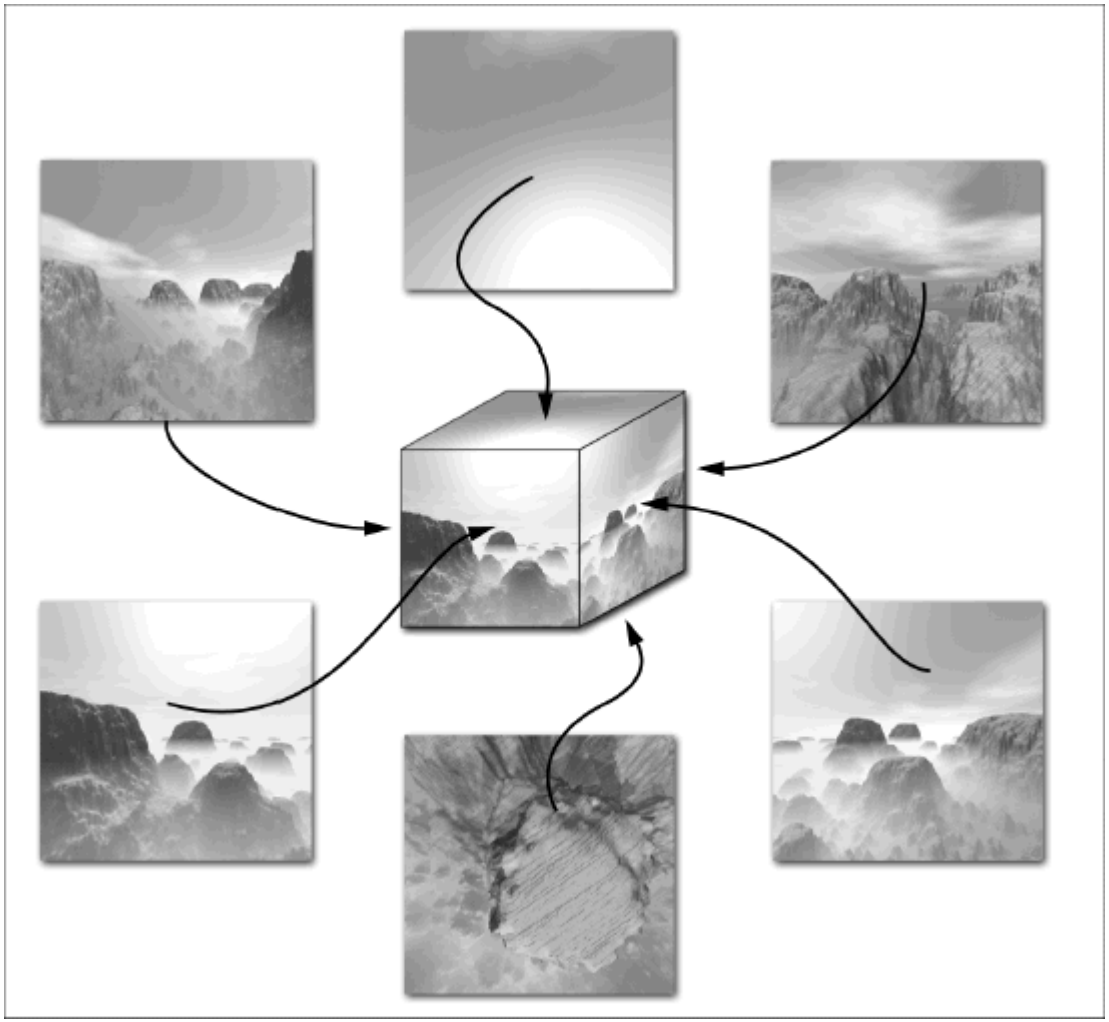
- Toate Unitatile de Prelucrare Grafica ale placilor video recente suporta un tip de textura numita

  cube map (GL_TEXTURE_CUBE_MAP).

- Se folosesc 6 imagini textura care se asociaza feţelor cubului unitate centrat in originea sist de coord globale. Fiecare textura este adresata in spatiul 0<=u,v<=1.
- Cubul se foloseste ca suprafata intermediara pentru maparea in 2 pasi a imaginii mediului inconjurator pe obictele scenei 3D.
- Fiecare imagine a unei texturi cube map reprezinta una dintre cele 6 imagini panoramice ale mediului care inconjoara obiectele texturate: impreuna ele formeaza imaginea mediului inconjurator.
- Harta textura cube map se poate obtine creând imagini ale scenei, in cele 6 directii ale sistemului de coordonate carteziene (Ox, -Ox, Oy, -Oy, Oz, -Oz), dintr-un punct fix.
- Fiecare imagine trebuie sa aiba un câmp al vederii de 90 grade si forma unui patrat, astfel incat cele 6 fete ale cubului sa se poata imbina fara goluri sau suprapuneri.
- Metoda are multe aplicatii: **Skyboxes, Skylight Illumination, Global Illumination, Stable Specular Highlights.**

Cube Environment Mapping(2)

Harta textura



Cube Environment Mapping - implementare -

//Fragment shader

```
in vec3 viewDirection; //coord globale – directia observatorului
in vec3 normalDirection; //coord globale
uniform samplerCube cubeMap;

void main
{
	vec3 reflectedDirection = reflect(viewDirection, normalize(normalDirection));

	gl_FragColor = textureCube(cubeMap,  reflectedDirection);

	// vec3 refractedDirection = refract(viewDirection, normalize(normalDirection), n1/n2);

	// n1, n2 – indicii de refractie

	//gl_FragColor = textureCube(cubeMap, refractedDirection);

	}
```

Laboratorul 4

https://learnopengl.com/Advanced-OpenGL/Cubemaps

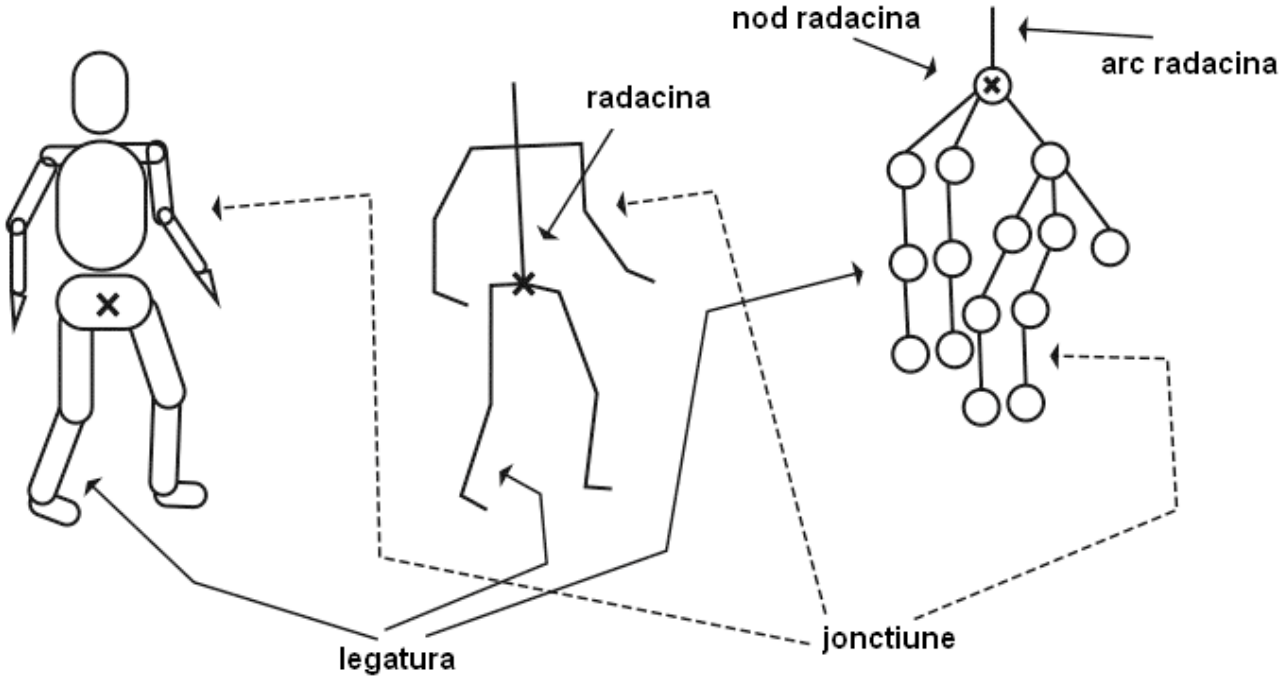
Modelarea si animatia structurilor articulate

Prof. univ. dr. ing. Florica Moldoveanu

Modelarea unei structuri ierarhice articulate(1)

- O structura articulata este alcatuita din:
  - obiecte rigide, numite legături
  - jonctiuni, in care sunt conectate obiectele rigide

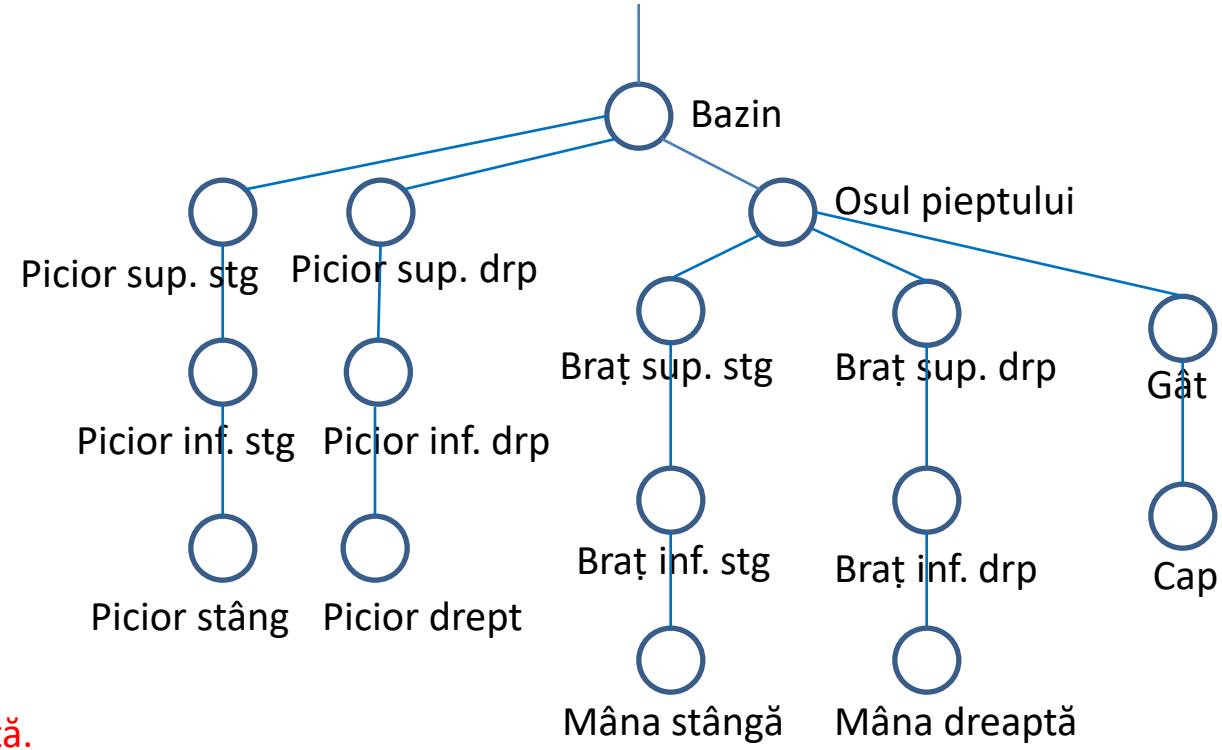
- Scheletul unui personaj animat poate fi modelat convenabil printr-o structura articulata:
  - legaturile reprezinta oasele
  - jonctiunele sunt articulatiile în care sunt conectate oasele



Structura articulata | Reprezentarea ierarhica | Arborele structurii ierarhice

Modelarea unei structuri ierarhice articulate(2)

Structura articulată simplificată.

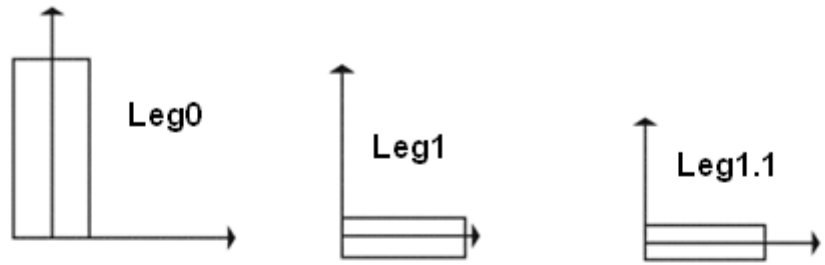


Reprezentarea ierarhică 2D a unui schelet uman

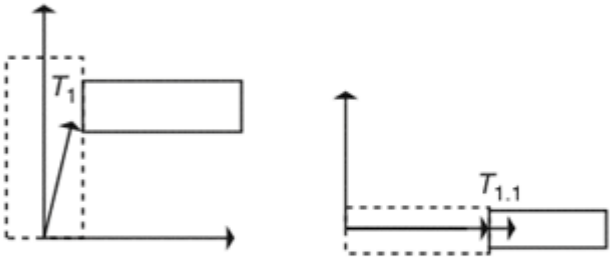
- Structura articulată se poate reprezenta printr-un arbore, in care:
  - nodurile reprezinta legaturile (oasele)
  - arcele reprezinta jonctiunile
- In arborele unui schelet uman radacina corespunde osului bazinului

Compunerea unei structuri articulate

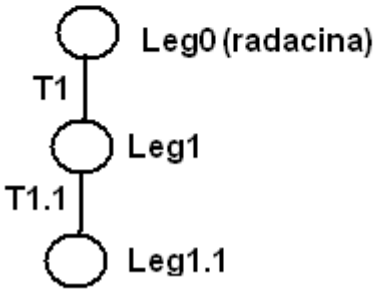
- Fiecare obiect rigid al structurii articulate este definit in propriul său sistem de coordonate.

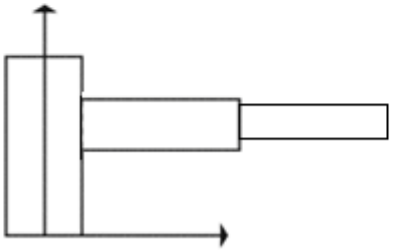


- Structura articulata este compusa aplicând fiecarui obiect component o transformare din sistemul său de coordonate locale in sistemul de coordonate locale al obiectului la care se conecteaza.



- Transformarea este atasata arcului din arbore care conecteaza nodul obiectului conectat cu nodul său parinte (nodul obiectului la care se conecteaza).



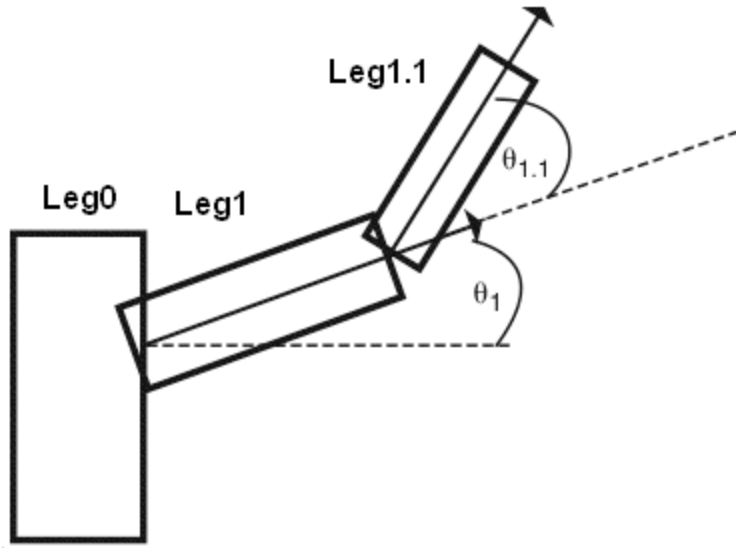


Structura articulata compusa, poza initiala

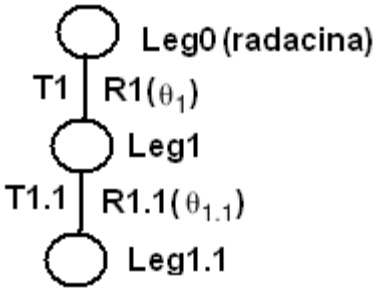
➢ Transformarile de compunere a structurii articulate nu se modifica in timpul animatiei!

Animarea structurii articulate (1)

- In timpul animatiei, fiecare legatura se poate roti in articulatia care o conecteaza de legatura aflata pe nivelul ierarhic imediat superior.

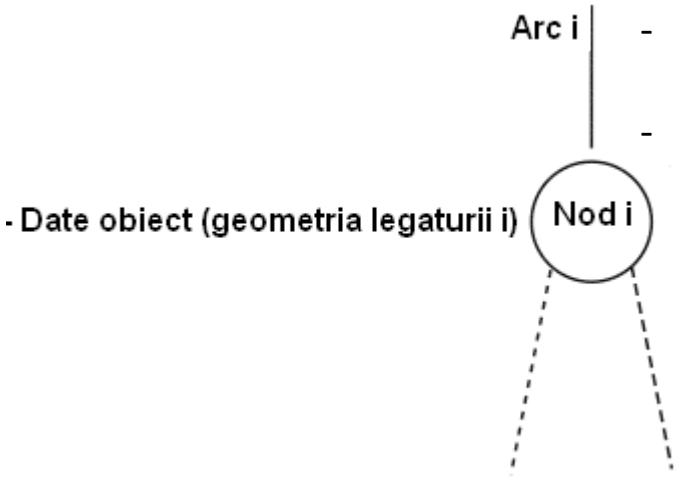


- Aceasta rotatie este cea de-a 2-a transformare atasata arcului care conecteaza nodul corespunzator legaturii rotite de nodul sau parinte.



Animarea structurii articulate (2)

- Fiecarui nod ii sunt asociate datele care descriu geometria legaturii (obiectului rigid) pe care o reprezinta. In mod simplificat, un os poate fi reprezentat printr-o poziție 3D, care corespunde unei articulatii, si un vector, care da lungimea si directia sa.



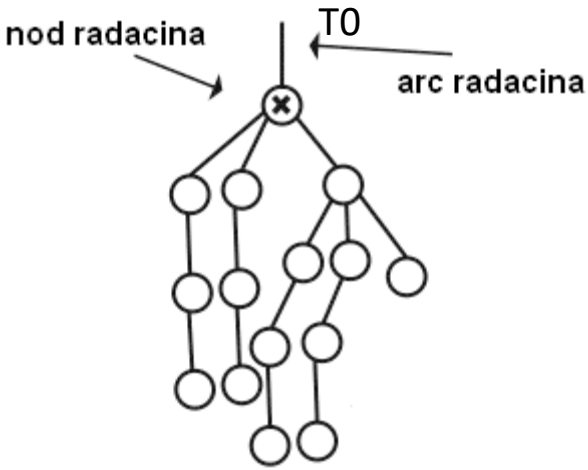
- Ti: Transformarea de compunere a legaturii i cu legatura (i-1) **constantă**
- Ri(u): Rotatia legaturii i în jonctiunea cu legatura (i-1) **variabilă**

T= Ti*Ri(u)

- Ri: rotatia leg i in jurul originii sist său de coord locale
- Ti: transfomarea de compunere a legaturii i cu leg. (i-1)

Animarea structurii articulate(3)

- Pozitia intregii structuri articulate in sistemul coordonatelor globale, la un moment dat, este definita printr-o transformare aplicata nodului radacina: T0.
- Transformarea nodului radacina, T0, este atasată arcului radacina: schimbarea acestei transformari va repozitiona rigid intreaga structura, in sistemul coordonatelor globale.
- Pozitiile celorlalte noduri ale ierarhiei sunt localizate relativ la pozitia nodului radacina.



- Prin modificarea in timp a transformarii T0 şi a rotațiilor ataşate arcelor arborelui se modifica pozitia si orientarea tuturor elementelor structurii articulate rigide.

Transformarea vârfurilor structurii articulate

- Pentru afişarea structurii articulate, se transforma vârfurile fiecarei legaturi din sistemul coordonatelor locale în sistemul coordonatelor globale.

- T0 este transformarea care pozitioneaza intreaga structura in sist. coordonatelor globale.

- Varfurile legaturii leg0

  V0' = T0•V0 = A0•V0

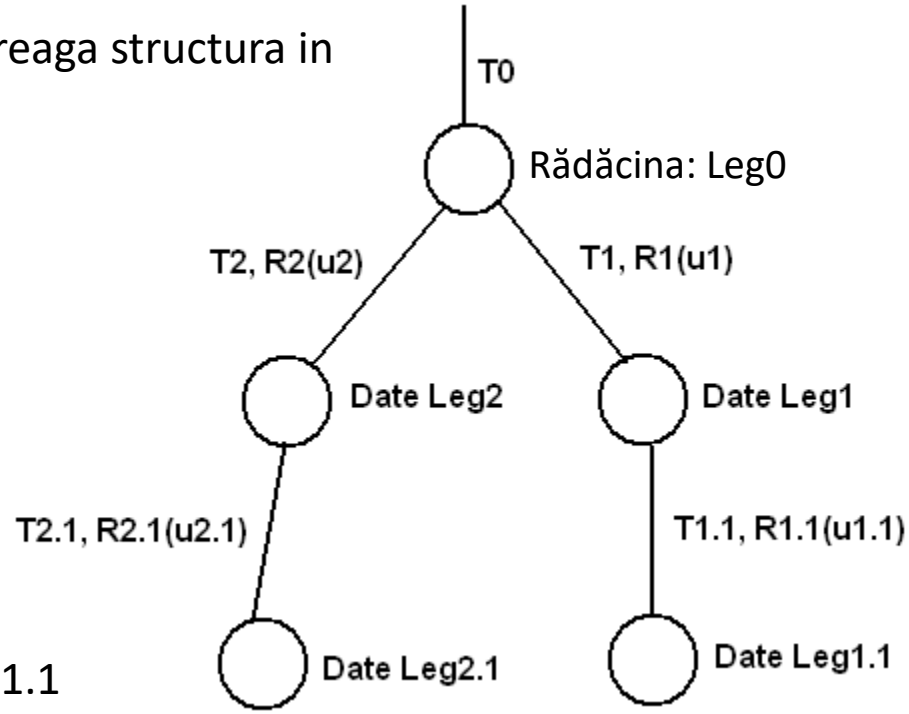
-Varfurile legaturii Leg1

  V1' = T0•T1•R1(u1)•V1 = A1•V1

-Varfurile legaturii Leg1.1

  V1.1' = T0•T1•R1(u1)•T1.1•R1.1(u1.1)•V1.1

  = A1.1•V1.1



➢ Ai reprezinta matricea de transformare din coordonate locale în coordonate globale a varfurilor legaturii i, la un moment al animatiei.

Animatia prin interpolare intre cadre cheie (1)

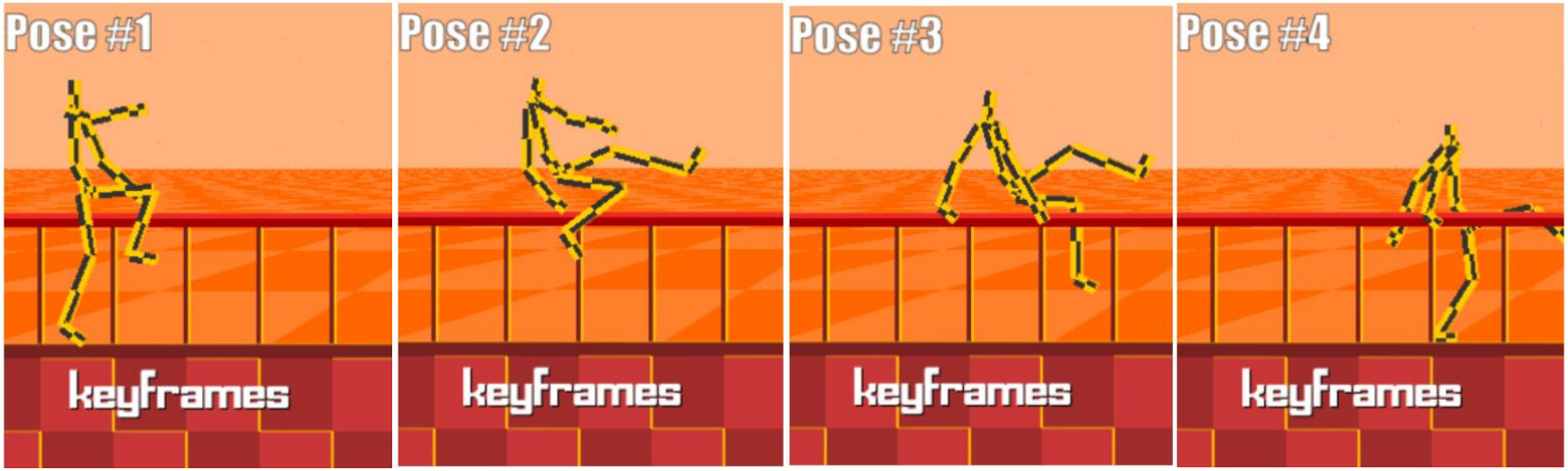
- Un set complet de parametri care redau structura articulata intr-o pozitie spatiala, se numeste ***poză***.
- O poză se reprezintă printr-un ***vector poză***, care contine cate un parametru pentru fiecare joncțiune (arc din arborele structurii articulate): unghiul de rotație al unei legaturi în joncțiunea cu legatura parinte.

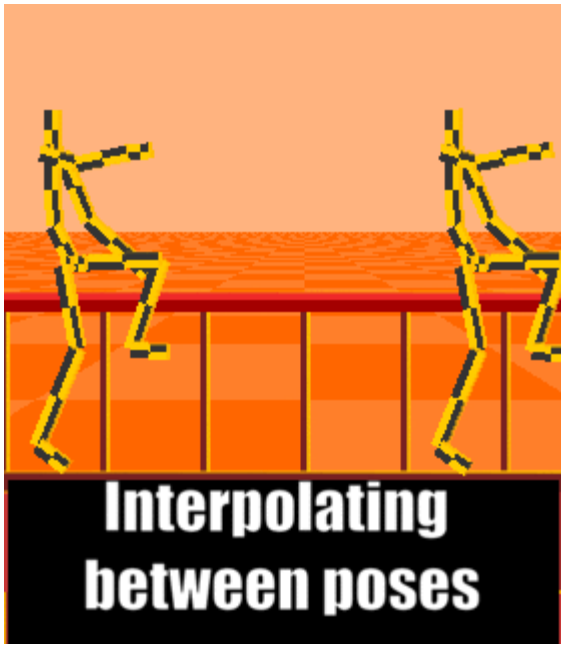
- Animatorul defineste un set de "poze cheie", numite si "cadre cheie"
- Cadrele intermediare dintre cadrele cheie se obtin prin interpolarea parametrilor ataşați jonctiunilor.

Exista 2 metode de calcul al pozelor cheie:

- Cinematica directa
- Cinematica inversa

Animatia prin interpolare intre cadre cheie (2)





https://learnopengl.com/Guest-Articles/2020/Skeletal-Animation

Cinematica directa (Forward kinematics)

Pozitionarea unei structuri articulate prin specificarea parametrilor atasati tuturor arcelor (jonctiunilor) se numeste *cinematica directa*.

- Pentru a calcula o anumita pozitie si orientare pentru un obiect frunza al arborelui intr-un cadru cheie, animatorul trebuie sa calculeze unghiurile de rotatie atasate tuturor arcelor de la nodul frunza la radacina.

Definirea unei poze prin specificarea unghiurilor de rotatie atasate arcelor este o munca extrem de obositoare şi dificila.

De aceea, se poate recurge la ***cinematica inversa***.

Cinematica inversa (Inverse kinematics)(1)

- Cinematica inversa permite animatorului sa se concentreze numai pe conditiile pe care trebuie sa le indeplineasca obiectele din frunzele arborelui.
- Animatorul defineste:
  - poza initiala si
  - pentru fiecare obiect frunza din structura ierarhica (“end effector”)
    - pozitia si orientarea sa în poza finala
- **Pornind de la fiecare obiect frunza, sistemul de animatie trebuie sa calculeze valorile tuturor arcelor (jonctiunilor), de la nodul frunza pana la nodul radacina, astfel incat sa se obtina configuratia dorita pentru obiectul frunza.**
- Valorile calculate pentru toate jonctiunile reprezinta vectorul poza finala.

Cinematica inversa(2)

- **Problema calculului vectorului "poza finala"poate avea:**
  - **zero solutii** (sistemul este supra-constrans)**,**
  - **o solutie,**
  - **mai multe solutii** (sistemul este sub-constrans)**!**
- Solutia poate fi:
  - analitica, daca problema este simpla (lanturi foarte scurte)
  - numarica, iterativa – in caz contrar
- Pot fi definite expresii de control prin care se specifica preferinta pentru un anumit tip de solutie.

Modelarea si animatia personajelor (Character animation)

Prof. univ. dr. ing. Florica Moldoveanu

Animatia reprezentarii prin plasa poligonala(1)

- Plasa poligonala (mesh) a fost prima metoda de reprezentare a personajelor, direct inspirata din tehnicile de animatie folosite in plan. Ex: personajele din jocurile Quake.
- Personajul este reprezentat printr-o plasa poligonala 3D.
- Animatia personajului consta in deformarea plasei, fiecarui cadru imagine corespunzandu-i o “poza” (ipostaza) a personajului.
- Numarul de varfuri ale plasei si conectivitatea lor nu se modifica de la o poza la alta, ci doar pozitiile varfurilor.
- De regula, animatorul defineste doar un numar de « poze cheie », restul pozelor fiind obtinute prin interpolarea intre pozele cheie.

Animatia reprezentarii prin plasa poligonala (2)

- Exista multe aplicatii comerciale (de exemplu, 3D Studio Max) care permit editarea plasei poligonale in spatiul 3D, pentru definirea cadrelor cheie.
- Se pot genera diferite secvente de animatie ale unui personaj, care redau diferite stari sau actiuni ale personajului: plimbarea, alergarea, cautarea, atacul, etc. Aceste secvente de animatie pot fi preluate pentru a fi folosite in jocuri sau alte tipuri de aplicatii.

Avantajul metodei:

- Animatorul poate decide cat de apropiat de realitate sa fie modelul personajului. Teoretic, cu suficiente poligoane se poate obtine orice.

Dezavantajele metodei:

- Trebuie creat cate un model 3D (poza) pentru fiecare cadru cheie.
- Consumul de memorie creste odata cu numarul de poligoane ale modelului.
- Pozele din cadrele cheie trebuie sa fie foarte apropiate pentru a se obtine rezultate bune.

Animatia reprezentarii prin plasa poligonala (3)

Dezavantajele metodei (cont):

- Animatiile produse sunt greu de refolosit, ceea ce duce la costuri mari .
- Calitatea animatiei depinde de inspiratia si talentul animatorului.

  Spre deosebire de o animatie 2D, o animatie 3D este mult mai greu de conceput: trebuie sa arate cat mai natural din orice pozitie 3D ar fi privita.
- Volumul mare de munca necesar pentru a produce o animatie si implicarea puternica a omului ii limiteaza aplicatiile.
- Nu se pot crea animatii care sa se poata adapta unor conditii cunoscute de-abia la rulare. Pentru aceasta ar fi necesar sa se creeze un numar foarte mare de animatii.

Metoda poate fi implementata eficient folosind Unitatea de Prelucrare Grafica: interpolarea intre cadrele cheie poate fi efectuata in timp real de GPU, nemaifiind necesara generarea anterioara si stocarea cadrelor intermediare.

Animatia bazata pe schelet

- Animatia bazata pe schelet (“skeletal animation”) - metoda standard de realizare a animatiei personajelor.
- Personajul este reprezentat prin doua sau mai multe “straturi”(reprezentare “multistrat”):
  - Schelet + piele
  - Schelet + muschi + piele
- Animatia se transmite de la schelet catre straturile superioare.

- Exista multe aplicatii comerciale care permit definirea personajelor si a secventelor de animatie pe baza acestei metode.
- Este larg utilizata in jocuri si industria de filme; poate fi aplicata nu numai personajelor ci si oricaror obiecte alcatuite din elemente rigide articulate.

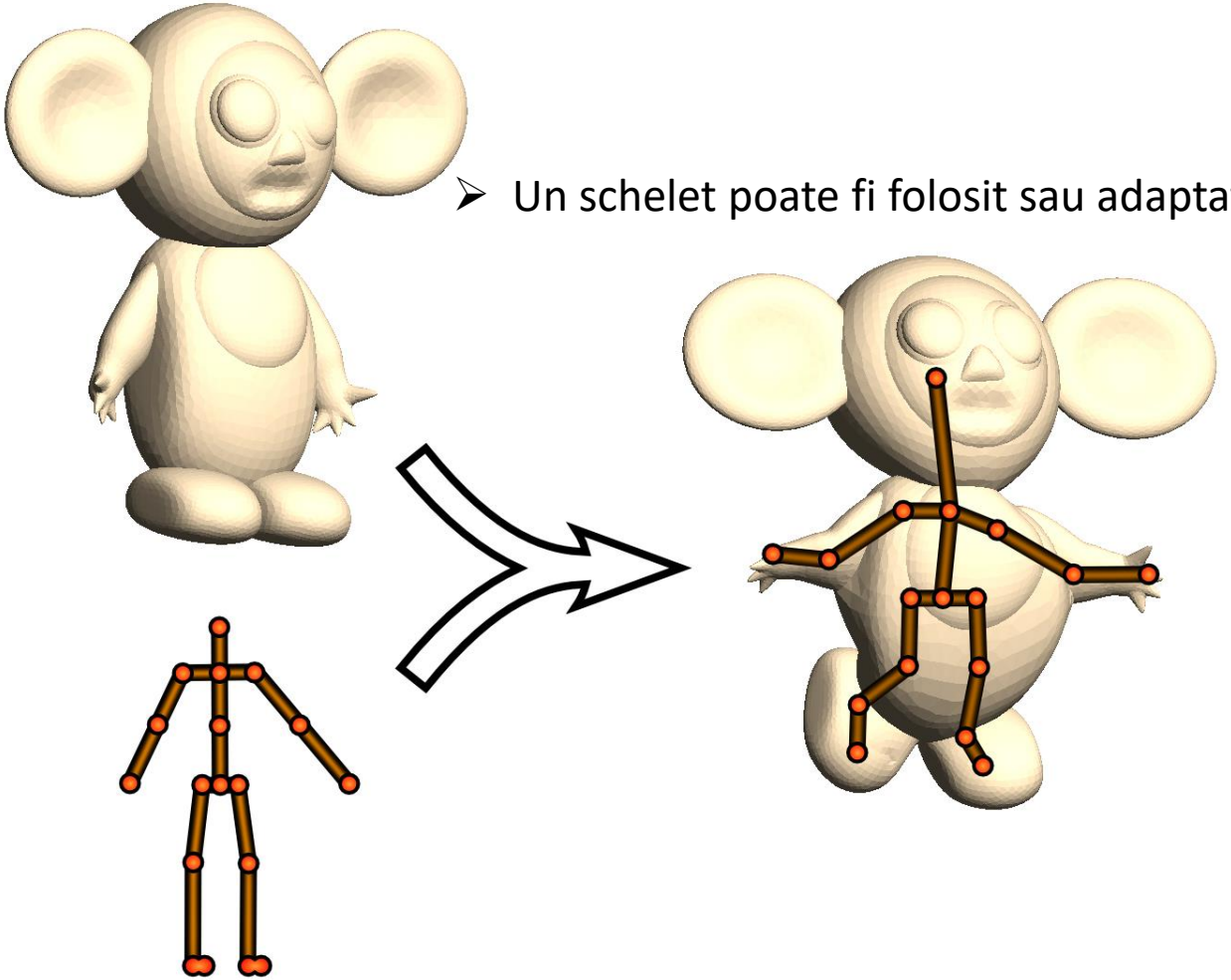
- “Skinning”, sau SSD (Skeleton Subspace Deformation), este o metoda populara de animare a personajelor modelate prin schelet si piele:
  - Scheletul este reprezentat ca o structura ierarhica articulata
  - Pielea este reprezentata printr-o plasa poligonala (mesh), atasata oaselor scheletului

Skinning (1)

- In sistemele de animatie, **pielea este reprezentata printr-un obiect “mesh”, asociat unui schelet**.
- **Scheletul este doar un mecanism de control al deformarii obiectului mesh atasat**. Este reprezentat printr-o structura ierarhica articulata in care oasele sunt “noduri nule”: oasele nu au geometrie.
- Un os poate fi reprezentat printr-o locatie 3D, care corespunde unei articulatii, si un vector, care da lungimea si directia sa.
- Sistemele de animatie vizualizeaza in diferite moduri scheletul. De exemplu Maya si Okino vizualizeaza articulatiile prin mici sfere, fiecare os conectand o sfera de centrul urmatoarei sfere din ierarhie .

- “Pielea” este atasata unui schelet in sistemul de animatie, iar operatia se numeste “rigging”: fiecare varf al mesh-ului piele este asociat unuia sau mai multor oase
- Pentru animatie, fiecare varf al « pielii » este transformat pe baza transformarilor oaselor asociate.

Skinning (2)



- Un schelet poate fi folosit sau adaptat pentru mai multe mesh-uri:

Automatic Rigging and Animation of 3D Characters

Ilya Baran, Jovan Popovic

https://www.cs.toronto.edu/~jacobson/seminar/baran-and-popovic-2007.pdf

Skinning (3)

- Transformarea unui varf al mesh-ului poate fi influentata de unul sau mai multe oase.
- Numarul de oase la care este atasat un varf este limitat la 4 (practic, nu este nevoie de mai multe)

http://graphics.stanford.edu/courses/cs468-05-fall/Papers/p399-james.pdf

Vârfurile colorate: ataşate unui singur os

Vârfurile negre: ataşate mai multor oase – amplasate în aproprierea încheieturilor

Skinning (4)

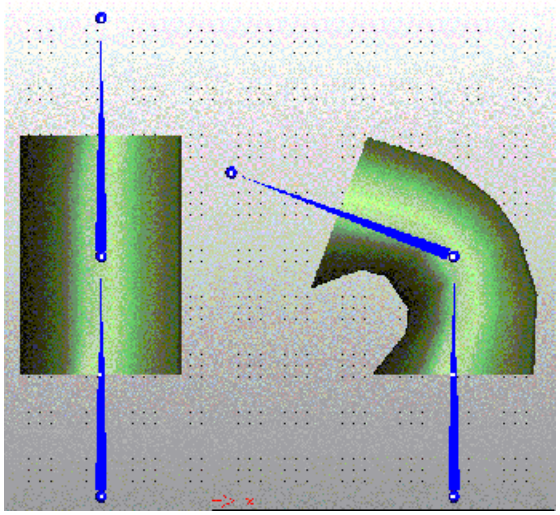
- Fiecare os contribuie la transformarea unui varf cu o anumita pondere.

w_{ij} - ponderea asociata varfului V_i pentru fiecare Os_j

daca $w_{ij} = 1$ → varful V_i este atasat rigid osului Os_j

Este necesar ca:

- Ponderile sa fie pozitive
- Suma ponderilor w_{ij} pentru un varf sa fie egala cu 1.



In figură:

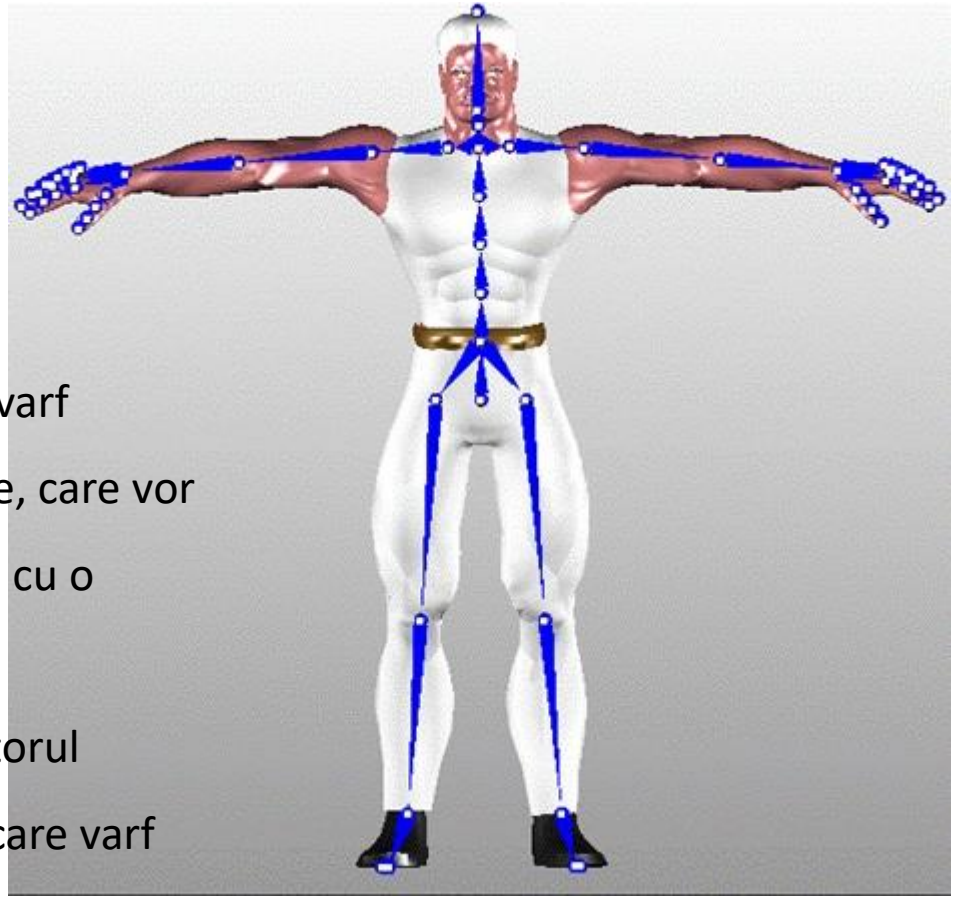
• Plasa poligonala cilindrica este deformata cu ajutorul unui schelet format din 2 oase.

• Varfurile plasei sunt atasate cu ponderi diferite celor 2 oase.

Skinning (5)
Legarea pielii la schelet

Poza de legare a pielii la oase

- Plasa poligonala este mai intai "*legata*" de ierarhia de oase, rezultand "*poza de legare*", numita si "*poza de referinta*".



http://www.okino.com/conv/skinning.htm

- Prin aceasta operatie, se asociaza fiecarui varf netransformat al pielii unul sau mai multe oase, care vor influenta pozitia sa in timpul animatiei, fiecare cu o anumita pondere.
- *Legarea* este efectuata de animator cu ajutorul sistemului de animatie, specificand pentru fiecare varf oasele asociate si ponderea fiecarui os.
- Legarea pielii la oase se face intr-o pozitie initiala a scheletului, care arata, in general, ca in figura:

Skinning (6)
Legarea pielii la schelet

Sistemul de coordonate al pielii

- Pielea este modelata intr-un sistem de coordonate local convenabil: de regula, un sistem de coordonate 3D dreapta, cu axa Z orientata catre observator.
- **Legarea (binding) pielii la schelet se face "potrivind" fiecare os al scheletului in plasa poligonala.**
- Fiecare os al scheletului este definit in propriul sau sistem de coordonate locale.
- Pentru legarea unui os la piele, i se aplica o transformare, numita ***transformare de legare*** a ***osului la piele***: transformare din sistemul de coordonate local al osului in sistemul de coordonate al pielii nedeformate (poza de legare). Tos_i - matricea transformarii de legare a osului i la piele - concateneaza toate transformarile (de compunere) atasate arcelor scheletului de la nodul i pana la radacina.
- Transformarea de legare a unui os la piele nu se modifica pe parcursul animatiei.
- Pe parcursul animatiei, fiecare poză a structurii articulate de oase este reprezentata prin **matricile de transformare a oaselor** din sistemele lor de coordonate locale in sistemul coordonatelor globale (calculate prin cinematica directa sau inversa). **Aceste matrici se folosesc la transformarea varfurilor plasei care modeleaza pielea.**

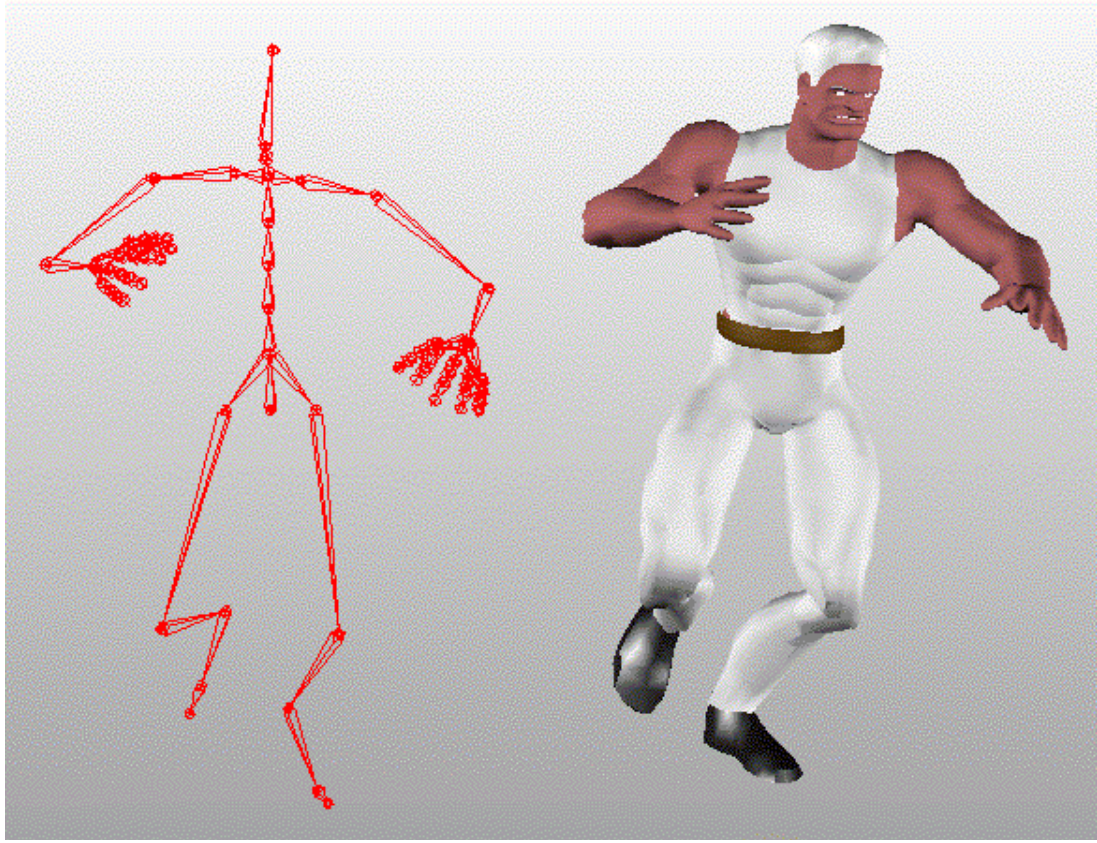
Skinning (7)
Animatia

- In fiecare cadru al animatiei se calculeaza varfurile plasei poligonale astfel:

$V' = \Sigma\, w_i * Aos_i * Tos_i^{-1} * V_{mod}$

cu $\Sigma\, w_i = 1$, $i = 1, n$

- V_{mod} este pozitia varfului in sistemul de modelare al pielii
- w_i este ponderea cu care este atasat varful osului i,
- Tos_i este transformarea de legare a osului i la piele
- (Tos_i^{-1}) transforma varful in sistemul de coordonate local al osului i,
- Aos_i este matricea de animatie a osului i, care transforma osul din sist. coord. sale locale in sist. coord. globale.



http://www.okino.com/conv/skinning.htm

V' – coord. varfului in sistemul coordonatelor globale.

Skinning (8)
Animatia

- In timpul animatiei, matricile de legare ale oaselor sunt utilizate pentru transformarea fiecarui varf al plasei, din spatiul de modelare al pielii in sistemul de coordonate local al fiecarui os asociat.
- In continuare, varfului i se aplica transformarea de animatie a osului, cu ponderea asociata, rezultand coordonatele varfului in sistemul coordonatelor globale.
- Odata cu varfurile plasei trebuie transformate si normalele in varfuri, necesare in calculele de iluminare.
- Normalele sunt afectate numai de rotatiile cuprinse in matricile de transformare.

$$\mathbf{N}' = \Sigma\, w_i * R_i * (Ros_i^{-1})^T * \mathbf{N}_{mod}$$

- $\mathbf{N}_{mod}$ este normala în varful i, în spatiul de modelare al pielii
- Ros_i este partea superioara de 3x3 elemente a matricei Tos_i
- R_i este partea de rotatie din matricea Aos_i

Skinning (9)
Implementarea animatiei folosind numai GPU

- Metoda de animatie poate fi implementata in diferite moduri, repartizand sarcinile de prelucrare intre CPU si GPU.

I. Deformarea "pielii" efectuata numai de GPU

GPU efectueaza toate calculele necesare deformarii plasei poligonale prin care este reprezentata pielea, in timpul animatiei.

1. **Aplicatia calculeaza si memoreaza** coordonatele varfurilor plasei si normalele in varfuri, raportate la sistemul de coordonate local al fiecarui os:

$\mathbf{Vk}_i = Tos_i^{-1} * \mathbf{Vk}_{mod}$, i=1,2,3,4 $\qquad V' = \Sigma\, w_i * Aos_i * (Tos_i^{-1} * V_{mod})$

$\mathbf{Nk}_i = (Ros_i^{-1})^T * \mathbf{Nk}_{mod}$, i=1,2,3,4 $\qquad N' = \Sigma\, w_i * R_i * (Ros_i^{-1} * N_{mod})$

unde Vk_{mod}, Nk_{mod} reprezinta varful k, respectiv normala in varful k, in spatiul de modelare al pielii.

Skinning (10)
Implementarea animatiei folosind numai GPU

2. **Pe parcursul animatiei (pentru fiecare cadru), aplicatia transmite la GPU reprezentarile varfurilor (Vk_i) si ale normalelor (Nk_i) in cele 1,2,3 sau 4 sisteme de coordonate locale ale oaselor de care depinde fiecare varf.**

- Matricile de animatie ale oaselor, Aos_i, sunt accesibile programului varf ca parametri uniformi.

Transformarile realizate de GPU in vertex shader:

$\mathbf{Vk'} = \Sigma\, w_i \bullet Aos_i \bullet \mathbf{Vk}_i \quad i=1,2,3,4 \qquad \Sigma\, w_i = 1$

unde Aos_i este matricea de transformare a osului i ;

$\mathbf{Nk'} = \Sigma\, w_i \bullet R_i \bullet \mathbf{Nk}_i \quad i=1,2,3,4 \qquad \Sigma\, w_i = 1$

unde R_i este partea superioara de 3x3 elemente a matricei Aos_i (rotatia)

- Consum mare de memorie pe CPU

Skinning (11)
Implementarea animatiei folosind CPU + GPU

II. Deformarea "pielii" efectuata de CPU + GPU

1. Aplicatia calculeaza si memoreaza:

- Coordonatele varfurilor plasei in spatiul de modelare, **Vk**
- Normalele varfurilor plasei in spatiul de modelare, **Nk**
- Pentru fiecare varf, matricea Tos_i^{-1} prin care varful este transformat in sistemul de coordonate locale al osului i.

2. Pe parcursul animatiei (pentru fiecare cadru):

– aplicatia (CPU) calculeaza matricile **Ti** = $Aos_i \bullet Tos_i^{-1}$ care vor fi utilizate la transformarea varfurilor legate la osul i

– transmite la GPU (pentru Vertex Shader) varfurile si normalele in spatiul de modelare (**Vk, Nk)** precum si matricile **Ti.**

Skinning (12)
Implementarea animatiei folosind CPU + GPU

Matricile **Ti pentru toate cadrele animatiei** pot fi precalculate si memorate de aplicatie. Ele sunt accesibile programului varf ca parametri uniformi, fiind aceleasi pentru toate varfurile care depind de aceleasi oase.

Transformarile realizate de GPU:

$\mathbf{Vk'} = \Sigma\, w_i \bullet \mathbf{Tki} \bullet \mathbf{Vk}$ $\qquad \Sigma\, w_i = 1$

$\mathbf{Nk'} = \Sigma\, w_i \bullet \mathbf{R_{ki}} \bullet \mathbf{Nk}$ $\qquad \Sigma\, w_i = 1$

unde **Tki** sunt matricile Ti (i=1,2,3,4) de care depinde varful k

$\mathbf{R_{ki}}$ este partea de rotatie din matricea **Tki**

Programul varf prezentat in continuare implementeaza aceasta metoda.

Skinning (13)
Implementarea animatiei folosind CPU + GPU

Vertex shader

```
layout(location = 0) in vec3 pos;
layout(location = 2) in vec3 norm;
layout(location = 3) in vec2 tex;
layout(location = 6) in ivec4 boneIds;
layout(location = 7) in vec4 weights;

uniform mat4 projection;
uniform mat4 view;
uniform mat4 model;

const int Max_Bones = 100;
uniform mat4 BonesMatrices[Max_Bones]; //matricile de transf a oaselor pt cadrul curent

out vec2 TexCoords;
```

```
struct VertexFormat {
glm::vec3 Position;
glm::vec3 color;
glm::vec3 Normal;
glm::vec2 TexCoords;
glm::vec3 Tangent;
glm::vec3 Bitangent;
//indicii oaselor care influenteaza varful
int m_BoneIDs[4];//max 4 oase
//ponderea pt fiecare os
float m_Weights[4];
};
```

// Pentru o implementare completa: https://learnopengl.com/Guest-Articles/2020/Skeletal-Animation

Skinning (14)
Implementarea animatiei folosind CPU + GPU

Vertex shader

```
void main()
{   vec4 totalPosition = vec4(0.0f);
    for(int i = 0 ; i < 4 ; i++)
    { if(boneIds[i] == -1)
            continue;
      if(boneIds[i] >=MAX_BONES)
      {  totalPosition = vec4(pos,1.0f);
         break;
      }
      totalPosition += weights[i] * BonesMatrices[boneIds[i]] * vec4(pos,1.0f);
      vec3 localNormal = mat3(BonesMatrices[boneIds[i]]) * norm;
    }
   gl_Position =  projection * view * model * totalPosition;
    TexCoords = tex;
}
```

Animatia corpurilor rigide -1

Prof. univ. dr. ing. Florica Moldoveanu

Obiect rigid (Rigid body)

- Obiect care nu-si schimba forma, nici chiar daca asupra lui actioneaza o forta mare.
- Spatiul unui obiect rigid nu se intersecteaza niciodata cu spatiul unui alt obiect rigid.
- Un obiect rigid poate fi compus din mai multe parti. De ex. un personaj animat, reprezentat in mod simplificat prin obiecte geometrice conectate sau un schelet.

Obiectele rigide:

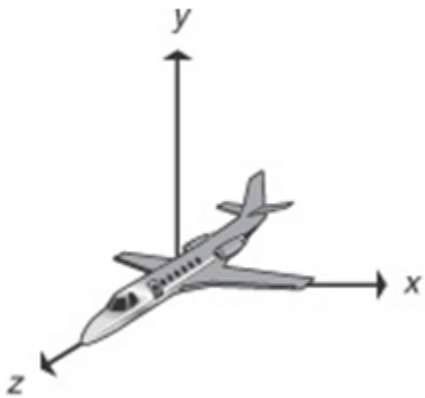
- Sunt opuse obiectelor deformabile, care-si modifica forma.
- Sunt o idealizare a realitatii, care conduce la simplificari ce permit simularea dinamicii unor astfel de corpuri.

Animatia unui obiect rigid

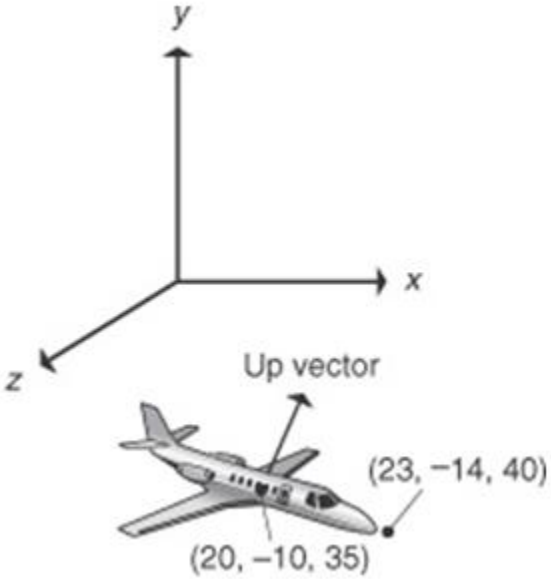
- Se poate reduce la o **modificare in timp a pozitiei si orientarii in spatiu** a fiecarei parti rigide din care este compus.
- Modificarea are la baza **interpolarea pozitiilor si a orientarilor între cadre cheie**
- **Simulari mai realiste ale dinamicii obiectelor rigide** se obtin tinand cont de legile fizicii, cum ar fi: inertia, acceleratia, gravitatia, frecarea, efectul coliziunilor, s.a.

Transformarea varfurilor obiectului (1)

- Obiectul rigid animat este descris prin coordonatele varfurilor raportate la un sistem de coordonate local, a carui origine reprezinta punctul de referinta al obiectului.
- In cazul unei structuri rigide articulate, fiecare element rigid este definit in propriul sistem de coord locale. Punctul de referinta al structurii rigide este pozitia radacinii structurii articulate.
- Pentru afisarea obiectului intr-un cadru al animatiei, coordonatele sale sunt transformate in sist. coord. globale.



Definitia obiectului in sistemul coordonatelor locale – spatiul obiect



Pozitia si orientarea obiectului, intr-un moment al animatiei, raportate la sistemul coordonatelor globale – spatiul scenei 3D.

Transformarea varfurilor obiectului (2)

- In timpul animatiei, punctul de referinta al obiectului se deplaseaza pe traiectoria miscarii.
- Pentru fiecare cadru cheie al animatiei, animatorul specifica:
  - Pozitia punctului de referinta al obiectului
  - Orientarea obiectului/Orientarea fiecarei legaturi rigide a structurii articulate in jonctiunea cu legatura parinte
- Pentru fiecare cadru intermediar dintre 2 cadre cheie:
  - Se calculeaza pozitia punctului de referinta pe curba traiectorie
  - Se calculeaza noile coordonate ale varfurilor obiectului, folosind:
    - Pozitia punctului de referinta
    - Orientarea/orientarile corespunzatoare cadrului, obtinute prin interpolarea orientarilor celor 2 cadre cheie
- Pentru fiecare cadru al animatiei, se transforma coordonatele varfurilor obiectului din sistemul coordonatelor locale in sistemul coordonatelor globale (al scenei 3D).

Specificarea animatiei(1)

- **Animatia consta in afisarea obiectului la diferite momente de timp, fiecare moment fiind asociat cu un cadru imagine.**
- Intre 2 cadre succesive, se pot modifica valorile mai multor parametri:
  - Coordonatele pozitiei punctului de referință al obiectului
  - Unghiul dintre 2 componente conectate ale obiectului (simularea structurilor articulate)
  - Transparenta
  - Culoarea sursei de lumina
  - Orice alt atribut utilizat in modelarea si redarea elementelor grafice
- **Valorile parametrilor de interes trebuie calculate pentru toate cadrele generate intre doua cadre cheie ale animatiei.**

Specificarea animatiei(2)

- **Interpolarea spatiala între 2 cadre cheie depinde şi de caracteristicile miscarii**.

De exemplu, intr-unul dintre cadrele intermediare obiectul se poate opri, apoi isi poate mari acceleratia, pentru a ajunge la acceleratia maxima intr-un alt cadru intermediar!

Metode de specificare a animatiei:

- Folosind limbaje de animatie de tip script.
- Folosind limbaje de programare (interpretate/compilate) simplificate, orientate catre artist.
- Folosind interfete grafice care genereaza script-uri.

Cerinte ale metodei de specificare a animatiei:

- Sa ofere o interfata grafica prietenoasa, usor de utilizat de animatori.
- Sa nu limiteze posibilitatile de descriere a animatiei.
- Sa permita memorarea animatiilor in scopul reutilizarii .

Specificarea animatiei (3)

Problema specificarii animatiei poate fi descompusa in 3 subprobleme:

1. Specificarea traiectoriei deplasarii: calea pe care se deplaseaza punctul de referinta al obiectului.
2. Specificarea caracteristicilor miscarii: viteza, acceleratia, aspecte temporale ale deplasarii obiectului pe traiectoria specificata.
3. Specificarea orientarii obiectului in timpul animatiei.

Animatorul poate modifica in mod independent cele trei componente ale specificatiei.

Specificarea traiectoriei deplasarii

- Traiectoria poate fi specificata printr-o curba de tip particular (circulara, spirala, sinusoidala) sau o curba de forma libera, definita in mod interactiv de catre animator.

Specificarea traiectoriei deplasarii(cont)

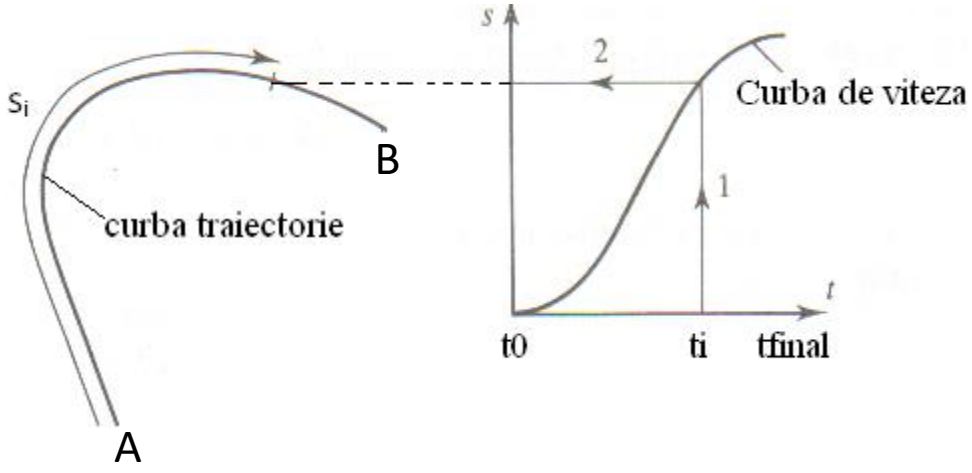
- Curba este 3D→ dificultati de percepere a tri-dimensionalitatii; solutii:
  - redarea simultana a proiectiilor traiectoriei in planele principale;
  - vizualizarea curbei traiectorie in scena 3D impreuna cu linii verticale de la ea in planul XOY.
- **Cazul general**: curba traiectorie este o curba parametrica 3D, Q(u) = (X(u), Y(u), Z(u)):
  - **o curba de interpolare**: Hermite(necesita tangente), Catmull-Rom (**preferata**-numai pozitii de pe curba)
  - **o curba de aproximare** (Bezier, B-spline)

- **Continuitatea**: apare in cazul curbelor compuse: Hermite, Bezier; este suficienta continuitatea geometrica de ordinul 1 (coliniaritatea tangentelor in punctul de jonctiune).
- **Controlul local** (preferat) / **global al curbei: B-spline, curbe compuse.**

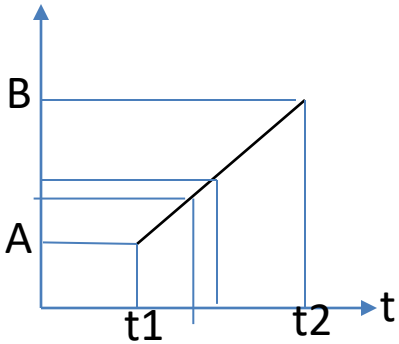
Specificarea caracteristicilor mişcării
Curbe de viteza (1)

Curba de viteza:

- Coreleaza timpul cu distanta parcursa de obiect pe curba traiectorie.
- Distanta este masurata prin lungimea arcului de curbă parcurs dintr-un punct de start, A, până la un moment de timp t_i.



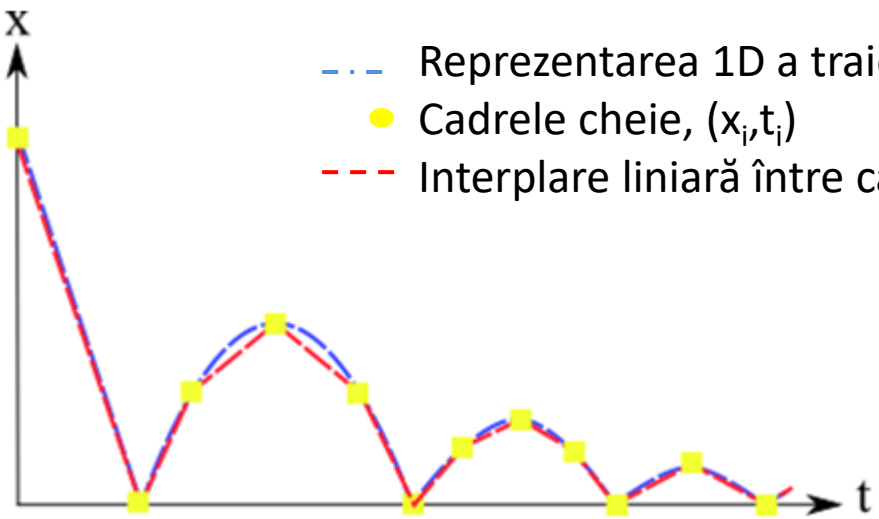
Specificarea caracteristicilor mişcării
Curbe de viteza (2)



Deplasare cu viteza constantă

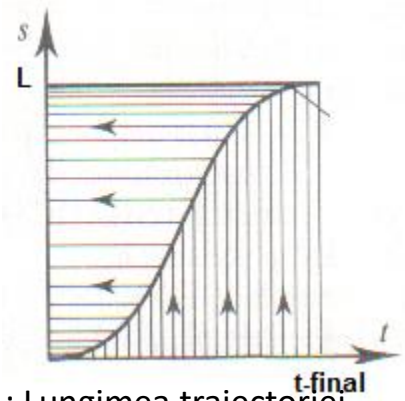
Exemplu: o maşină deplasandu-se cu viteza constanta între 2 puncte ale traiectoriei.

- Distanțele pe traiectorie dintre pozițiile corespunzătoare cadrelor intermediare sunt egale.
- Cu cât numărul de cadre intermediare este mai mare cu atât viteza de deplasare este mai mica



Reprezentarea 1D a traiectoriei unei mingi care sare, x(t).
Cadrele cheie, (x_i,t_i)
Interplare liniară între cadre cheie

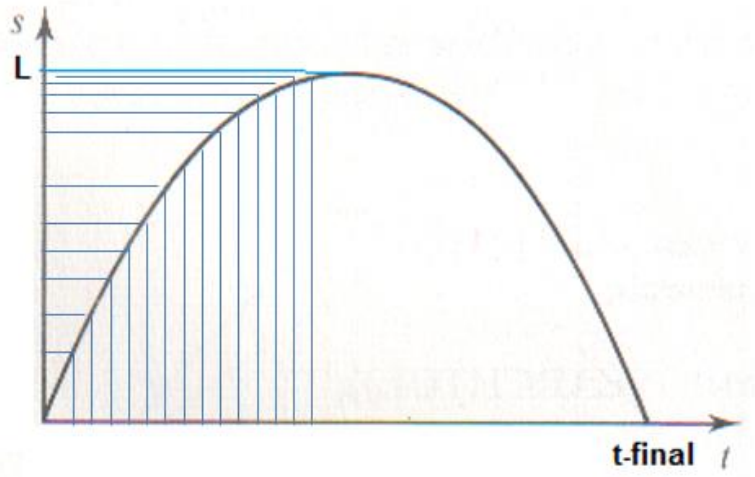
Specificarea caracteristicilor mişcării
Curbe de viteza (3)



L: Lungimea traiectoriei.

Curba "ease-in, ease-out" :
-acelerare lina la plecare
-scadere lina a acceleratiei la oprire

Se vor genera mai multe cadre intermediare la începutul şi la sfârşitul intervalului de timp (poziții apropriate pe traiectorie: deplasare lenta) şi doar câteva în mijlocul intervalului (pozitii indepartate pe traiectorie: deplasare rapidă).

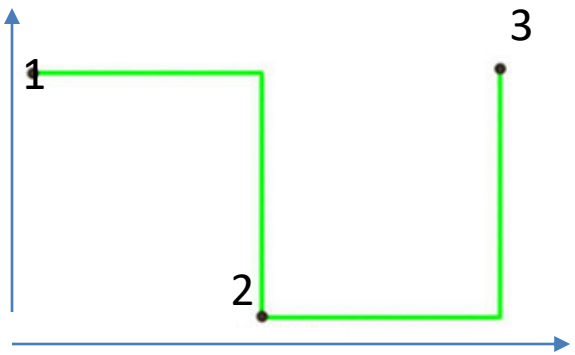


← Obiectul pleaca din punctul de start cu o viteza diferita de zero, care se micsoreaza treptat, pana la zero, atunci cand obiectul a ajuns la capatul traiectoriei, apoi incepe sa se deplaseze inapoi catre punctul de start, marindu-si viteza.

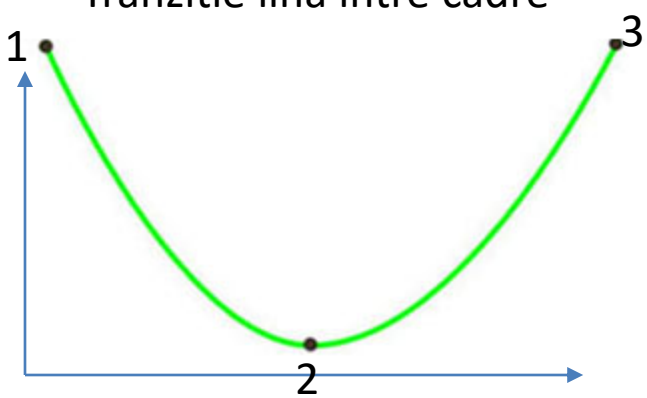
Specificarea caracteristicilor mişcării
Curbe de viteza- tranzitii intre cadre cheie

1-2 curba de viteza pt interpolarea intre cadrele 1-2

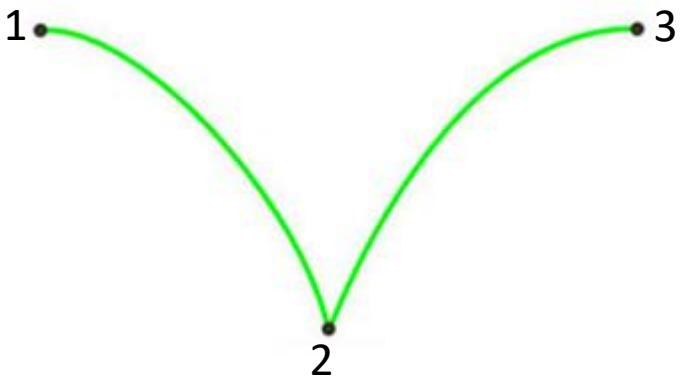
2-3 curba de viteza pt interpolarea intre cadrele 2-3



Tranzitie lină intre cadre



Tranzitie controlata prin tangente



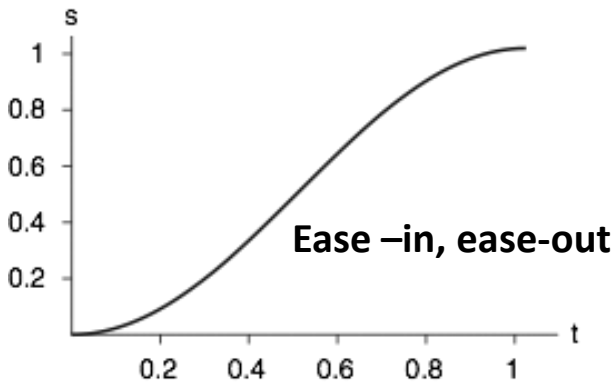
Specificarea caracteristicilor mişcării
Functii distanță-timp

- **Variante ale curbelor de viteza, s(t):**

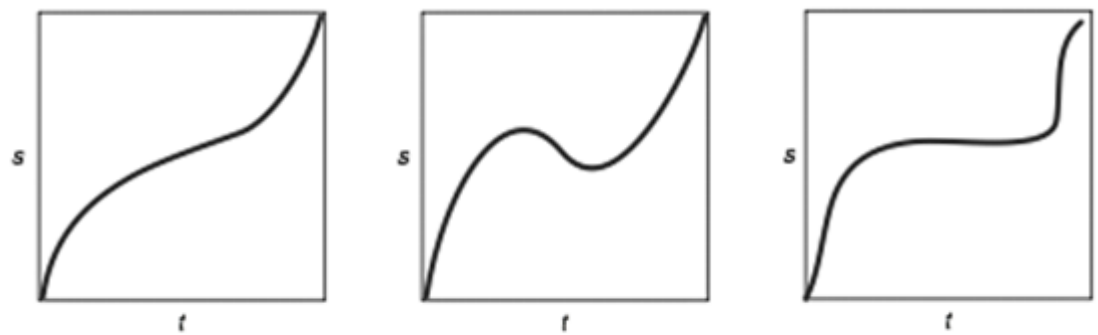
s: lungimea normalizata a arcului parcurs pana la momentul t, t ϵ [0,1], lungime arc ϵ [0,1]

lungimea normalizata a arcului de curba = lungime arc/lungimea totala a curbei;

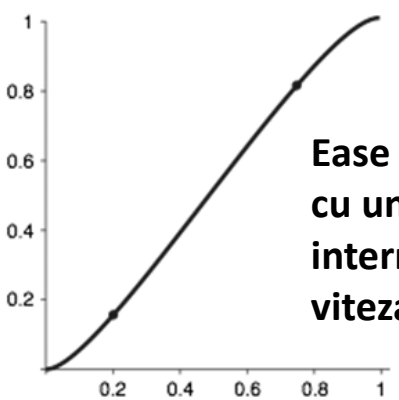
- Usor de definit in mod interactiv



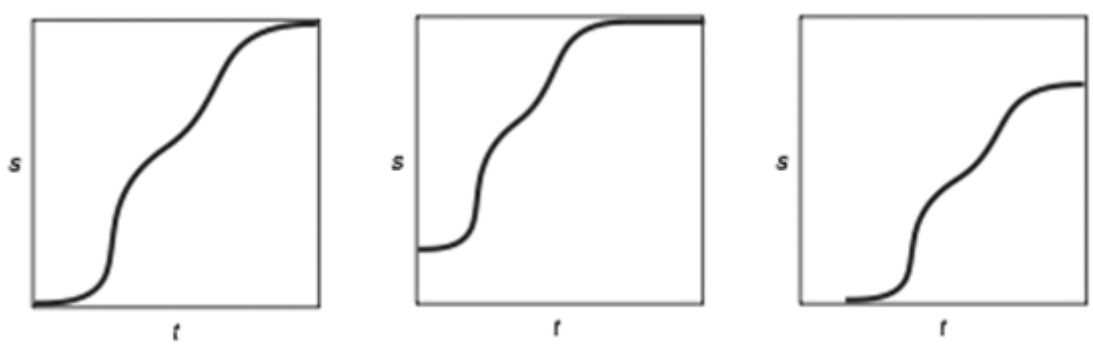
Ease –in, ease-out



pornire si oprire abrupte intoarcere oprire



Ease –in, ease-out cu un interval intermediar de viteza constanta



Specificarea caracteristicilor mişcării
Curbe de viteză

Avantajele metodei:

- Curbele de viteza sunt independente de traiectorie
- Animatorul poate specifica separat curba traiectorie si curba de viteza.
- Curbele de viteza sunt intuitive!
- Specificarea este usor de efectuat de animator: nu necesita cunostinte de Informatica.
- Exista curbe de viteza tipice pentru anumite caracteristici ale deplasarii: pot fi memorate pentru a fi reutilizate cu diferite traiectorii.

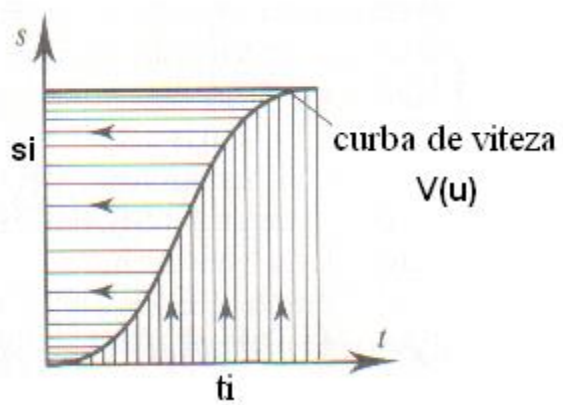
Probleme de calcul

- Curbele de viteza sunt curbe de forma libera 2D: $V(u) = (S(u), T(u))$.
- **Trebuie calculata pozitia pe traiectorie a obiectului la fiecare moment de timp al animatiei**

Corelatia curbelor traiectorie si viteza(1)

-Curba de viteza este parametrizata in functie de u, nu de t!

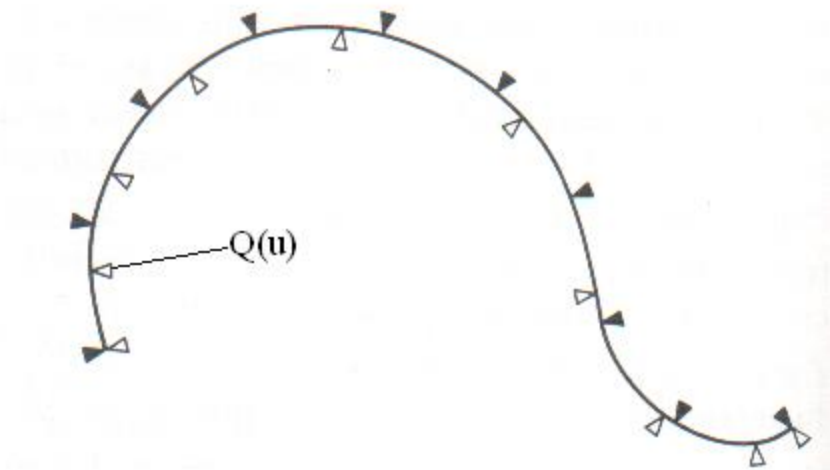
-Curba traiectorie este parametrizata in functie de u, nu de s - lungimea arcului de curba parcurs pana la momentul t!



Re-parametrizarea curbei de viteza in functie de t:

V(u) =(S(u), T(u)), 0<= u <=1

Re-parametrizarea curbei traiectorie in functie de s



←La intervale egale ale spatiului parametric (du) nu corespund arce de curba egale!

←Curba Q ar trebui re-parametrizata in functie de lungimea arcului de curba parcurs pana la un moment ti.

Corelatia curbelor traiectorie si viteza(2)

Curba de viteza: V(u)

- Este o curba parametrica 2D
- V(u) = (S(u), T(u)), T(u) este o functie strict crescatoare!

$$S(u) = a_{3s}*u^3 + a_{2s}*u^2 + a_{1s}*u + a_{0s}$$
$$T(u) = a_{3t}*u^3 + a_{2t}*u^2 + a_{1t}*u + a_{0t}$$

Corelatia curbelor traiectorie si viteza

Pentru fiecare moment de timp, ti, tinitial <= ti <= tfinal

- **se determina ui, a.i. T(ui) = ti**
- **se calculeaza si = S(ui)**, lungimea arcului de curba parcurs pana la momentul ti
- **se determina uk, astfel incat lungimea arcului de curba de la Q(0) la Q(uk) = si**
- **se calculeaza pozitia pe curba traiectorie la momentul ti: X(uk), Y(uk), Z(uk)**

Corelatia curbelor traiectorie si viteza (3)

Calculul lungimii arcului de curba parcurs pana la un moment de timp dat

O metoda aproximativa:

- Se calculeaza si se memoreaza intr-o tabela pozitii succesive de pe curba V(u), esantionand u cu pas mic:

u	t= T(u)	s= S(u)
0	t0 = T(0)	s0 = S(0)
du	t1 = T(du)	s1 = S(du)
2*du	t2 = T(2*du)	s2 = S(2*du)
3*du	t3 = T(3*du)	s3 = S(3*du)
.....	.....	.....

tk nu corespund momentelor discrete de timp ale animatiei.

Pentru fiecare moment de timp ti al animatiei:

- Se cauta in tabelă tk, tk+1, astfel incat: tk< ti < tk+1
- Se aproximeaza si interpoland liniar intre sk si sk+1

Specificarea şi interpolarea orientării

- Reprezentarea orientării în cadrele cheie:
  - —Sa fie usor de specificat
  - —Sa permita obtinerea unei secvente unice de orientari interpolate intre 2 cadre cheie

- **Posibilitati de reprezentare a orientarii**

1. Reprezentarea prin unghiuri de rotatie in jurul axelor sistemului de coordonate globale.
2. Rotatia in jurul axelor unui sistem de coordonate local
3. Reprezentarea printr-o axa si un unghi de rotatie in jurul axei
4. Reprezentarea prin quaternioni

Specificarea şi interpolarea orientarii
Rotatia in jurul axelor sistemului de coordonate globale (1)

- Transformarea care se aplica varfurilor obiectului pentru obtinerea unui cadru consta din una sau mai multe **rotatii in jurului axelor sistemului de coordonate locale**, **urmate de o translatie**: **este o transformare rigida**.
- Rotatia corespunzatoare unui cadru= produsul matricilor de rotatie in jurul celor 3 axe:

  R = Rx(ux)*Ry(uy)*Rz(uz) diferit de R' = Ry(uy)*Rx(ux)*Rz(uz)

Avantajele reprezentarii rotatiilor prin matrici de rotatie

- Usurinta in folosire: se cunosc foarte bine operatiile cu matrici si se pot implementa destul de usor folosind biblioteci standard.
- Rotatia poate fi combinata cu translatia prin inmultirea matricilor, si cu alte transformari, atunci cand transformarea totala este nerigida.

Specificarea şi interpolarea orientarii
Rotatia in jurul axelor sistemului de coordonate globale (2)

Dezavantaje ale reprezentarii prin matrici de rotatie

- Ordinea rotatiilor in jurul axelor este importanta. Rotatiile nu sunt independente.
- **Pentru animator este greu de determinat unghiul de rotatie in jurul unei axe, dupa o rotatie in jurul altei axe.**
- **Efectul "Gimbal Lock":** are loc atunci cand o serie de transformari de rotatie duce la pierderea unui grad de libertate de rotatie.

  Exemplu: fie R(a,b,c) matricea rotatiei in jurul axelor Ox, Oy, OZ:

$$R(a,b,c) = \begin{bmatrix} \cos(b)\cos(c) & \cos(c)\sin(a)\sin(b) - \cos(a)\sin(c) & \cos(a)\sin(b)\cos(c) + \sin(a)\sin(c) & 0 \\ \cos(b)\sin(a) & \cos(\mathrm{a})\cos(\mathrm{c}) + \sin(\mathrm{a})\sin(\mathrm{b})\sin(\mathrm{c}) & \cos(\mathrm{a})\sin(\mathrm{b})\sin(\mathrm{c}) - \cos(\mathrm{c})\sin(\mathrm{a}) & 0 \\ -\sin(\mathrm{b}) & \cos(\mathrm{b})\sin(\mathrm{a}) & \cos(\mathrm{a})\cos(\mathrm{b}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Specificarea şi interpolarea orientarii
Rotatia in jurul axelor sistemului de coordonate globale (3)

- Daca b = π/2 (cos(b) =0, sin(b) = 1) atunci:

$$R(a,b,c)=\begin{bmatrix} 0 & \cos(c)\sin(a)-\cos(a)\sin(c) & \cos(a)\cos(c)+\sin(a)\sin(c) & 0 \\ 0 & \cos(a)\cos(c)+\sin(a)\sin(c) & \cos(a)\sin(c)-\cos(c)\sin(a) & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}=\begin{bmatrix} 0 & \sin(a-c) & \cos(a-c) & 0 \\ 0 & \cos(a-c) & \sin(a-c) & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matricea nu depinde de unghiurile a si c, ci de diferenta (a-c): s-a pierdut un grad de libertate.

- **Interpolarea intre doua orientari cheie nu este unica:** se interpoleaza independent unghiurile de rotatie in jurul fiecarei axe, netinandu-se cont de interdependenta dintre ele.

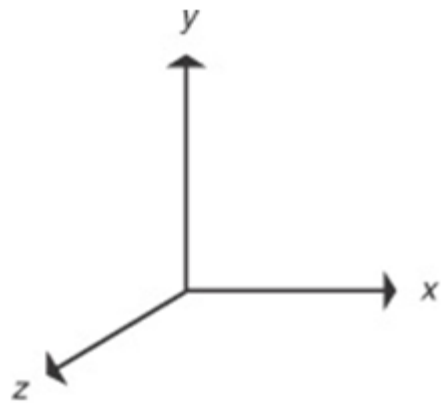
  Exemplu:

  - o rotatie cu π in jurul axei OX, Rx(π) , este echivalenta cu o rotatie compusa: R =Ry(π)*Rz(π)
  - la interpolare, rezulta secvente diferite (si imagini diferite):

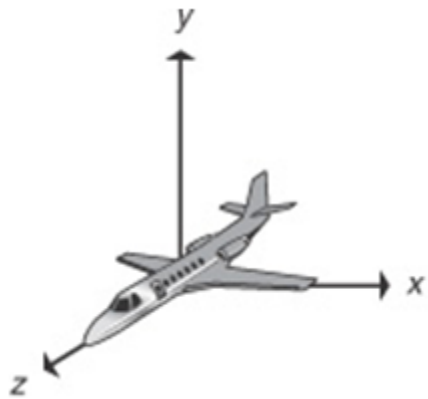
    a) R(0,0,0),……, R(πt, 0,0),……, R(π, 0, 0) 0<= t <= 1

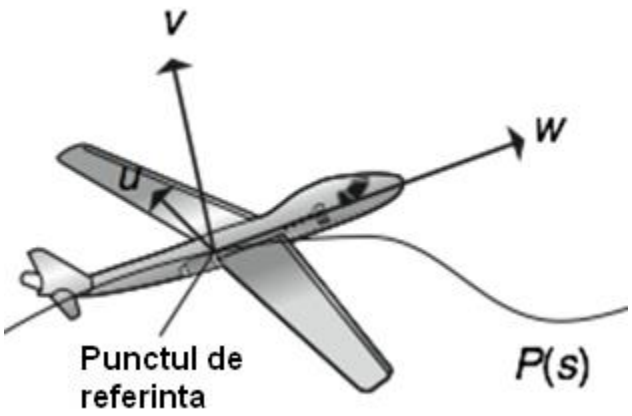
    b) R(0,0,0),……, R(0, πt, πt),……., R(0, π, π)

Specificarea şi interpolarea orientarii
Rotatia in jurul axelor unui sistem de coordonate local



Sistemul coordonatelor globale





Sistemul coordonatelor locale atasat obiectului

— La deplasarea pe traiectorie obiectul se deplaseaza odata cu sistemul său de coordonate local.

— Directia w este cea in care are loc deplasarea, axa v este “vectorul sus”, iar axa u rezulta din conditia ca (u,v,w) sa formeze un sistem de coordonate 3D dreapta.

—Modificarea orientarii axei w la deplasarea obiectului pe o traiectorie poate fi aleasa de animator; de ex: tangenta la traiectoria deplasarii.

—Orientarea pentru un cadru este specificata printr-o matrice care transforma sistemul de coordonate locale (x,y,z) al obiectului in cel corespunzator cadrului (u,v,w). Aceasta include rotatii in jurul axelor x,y,z: metoda are aceleasi dezavantaje ca rotatia raportata la un sistem de coord fix.

Specificarea şi interpolarea orientarii
Deplasamentul unghiular

Teorema Euler (1700): **o rotatie poate fi reprezentata in mod unic printr-o axa si un unghi R(ϕ, N), unde N este axa de rotatie (un vector 3D) si ϕ unghiul de rotatie in jurul axei N.**

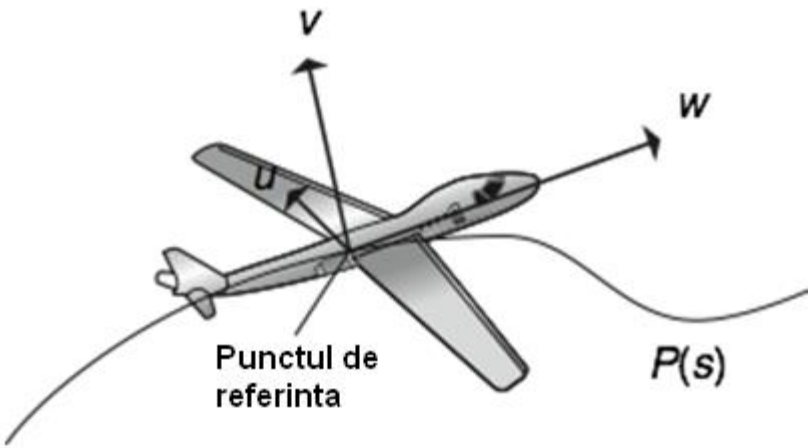
- Axa N poate fi directia w a sistemului de coordonate locale atasat obiectului iar ϕ unghiul de rotatie in jurul axei w.

Rezulta ca:

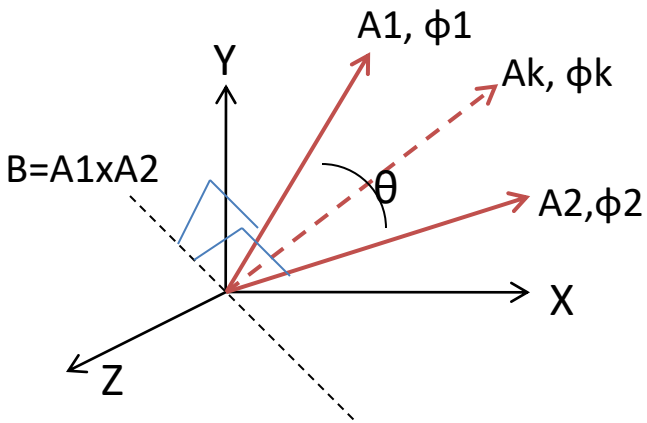
- **O orientare se poate reprezenta printr-o axa (diferita de la un cadru la altul) si un unghi de rotatie in jurul axei respective.**
- Interpolarea intre 2 orientari cuprinde:

  - interpolarea axelor de rotatie

  - interpolarea unghiurilor



Specificarea şi interpolarea orientarii
Interpolarea reprezentarii prin axa si unghi



Fie:
- A1 si A2 axele de rotatie pentru 2 cadre cheie succesive
- axa Ak, pentru cadrul intermediar k, interpolata intre A1 si A2.

$$\theta = \arccos(A1 \bullet A2/(|A1| * |A2|))$$

- Axa Ak se poate obtine rotind A1 catre A2 in jurul axei B, cu unghiul k*θ:

  Ak = Rotatie(B, k*θ, A1)

- Unghiul ϕk, pentru cadrul k, se obtine prin interpolare liniara intre unghiurile ϕ1 si ϕ2 :

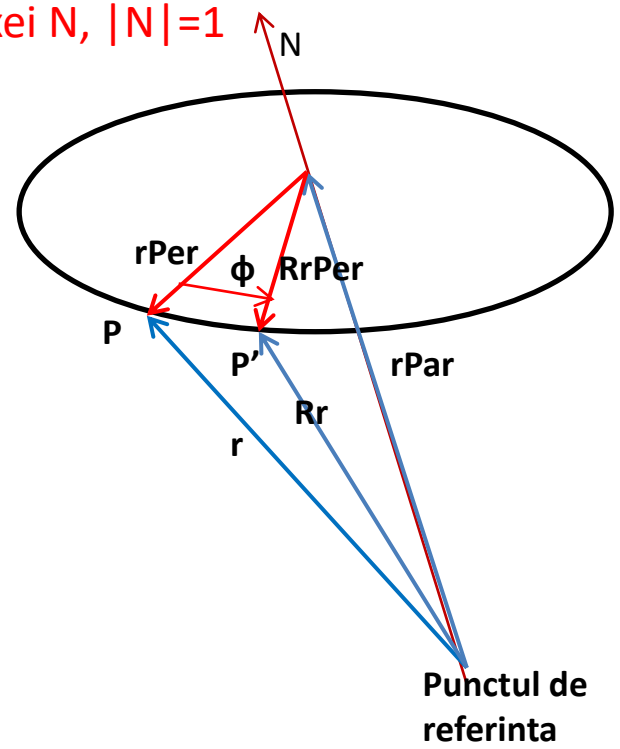
  $$\phi k = (1-k)*\phi 1 + k*\phi 2$$

Specificarea şi interpolarea orientarii
Rotatia unui varf in jurul unei axe

r : vectorul pozitie al unui varf, P, al obiectului animat, in sist. de coordonate locale

Rr: vectorul pozitie al varfului rotit, P', cu unghiul ϕ in jurul axei N, |N|=1

rPer si RrPer: vectori in planul perpendicular pe N, in care se afla P si P'



r = rPar + rPer
Rr = rPar + RrPer

rPar = k* N → rPer•N =0
rPer = r – rPar
(r – rPar) •N =0 → r •N – k*N •N = 0 → k = r •N

Rezulta:
rPar = (r •N)N; rPer = r – (r •N)N

Rrper = rPer * cos(ϕ) + V * sin(ϕ), unde V = N x rPer = N x r

Rr = (r •N)N + cos(ϕ) * (r – (r •N) N) + sin(ϕ) N x r = **r*cos(ϕ) + (1- cos(ϕ))(r•N)N + sin(ϕ) (N x r)**

Animatia corpurilor rigide -2

Prof. univ. dr. ing. Florica Moldoveanu

Specificarea şi interpolarea orientarii
Cuaternioni (1)

- Notiune matematica introdusa de matematicianul William Hamilton in 1843, in incercarea de a extinde planul numerelor complexe, a + ib ($i^2 = -1$) , pentru a defini un volum complex.
- A numit numarul

  q = a + bi + cj + dk ‘quaternion’

  cu proprietatea: $i^2 = j^2 = k^2 = -1$, ij = k; ji = -k
- Notatia prescurtata:

  q = (s,v) = s + v_x i + v_y j + v_z k, s este partea scalara (s – numar real) iar v partea vectoriala
  - (i,j,k) pot fi considerati ca versorii unui sistem de coord carteziene 3D iar (**v_x , v_y , v_z)** un vector raportat la acest sistem
- **In grafică se folosesc cuaternioni pentru a reprezenta orientarea prin axă si deplasament unghiular.**

Specificarea şi interpolarea orientarii
Cuaternioni (2)

- **Un punct P avand vectorul de pozitie r in sistemul de coordonate locale se reprezinta prin:**

  $p = (0, r)$ – **cuaternion pur**

- Deplasamentul unghiular cu ϕ – rotatia cu unghiul ϕ in jurul axei N - $R(\phi, N)$, se reprezinta prin **cuaternionul unitate** $q(s,v)$ ($s^2 + |v|^2 = |q|^2 = 1$) :

  $q = (\cos(\phi/2), \sin(\phi/2)*N), \quad |N| = 1$

- Rotatia punctului P in jurul axei N cu unghiul ϕ se exprima prin operatia

  $Rq(p)= q*p*q^{-1} = (0, \cos(\phi)*r + (1-\cos(\phi))*N*(N\bullet r) + \sin(\phi)*N \times r)$

  → la acelasi rezultat s-a ajuns prin calculul rotatiei unui punct in jurul unei axe (“Quaternions – optional”)

- **Compunerea rotatiilor se obtine prin inmultirea cuaternionilor:**

$q1 (s1, v1) * q2 (s2, v2) = (s1*s2 – v1\bullet v2, s1*v2 + s2*v1 + v1xv2)$

  - $\bullet$ reprezinta produs scalar iar x reprezinta produs vectorial

- Inmultind 2 cuaternioni de marime 1 se obtine un cuaternion de marime 1

Specificarea şi interpolarea orientarii
Avantajul reprezentarii orientarii prin cuaternioni

❖ **Interpolarea unică între 2 orientări reprezentate prin cuaternioni**

-**Folosind unghiuri Euler, o orientare se poate reprezenta prin rotatii diferite iar secventa de cadre obtinute prin interpolare intre cadre cheie nu este unica**:

- o rotatie cu π in jurul axei X, Rx(π) , este echivalenta cu o rotatie compusa: R = Ry(π)*Rz(π)

- la interpolare, rezulta secvente diferite:

a) R(0,0,0),……, R(πt, 0,0),……, R(π, 0, 0) 0<= t <= 1

b) R(0,0,0),……, R(0, πt, πt),……., R(0, π, π)

-**Folosind cuaternioni:**

-Rx(π) se reprezinta prin: (cos(π /2), sin (π/2)[1,0,0]) = **(0, [1, 0, 0])**

-Ry(π): **(0, [0,1,0])**

-Rz(π): **(0, [0,0,1])**

-Ry(π)*Rz(π) = (0, [0,1,0])x(0,[0,0,1]) = **(0, [1,0,0])**

-Concluzie: **cele 2 rotatii Euler echivalente se reprezinta prin acelasi cuaternion**

→ **interpolare unica intre 2 orientari reprezentate prin cuaternioni.**

Interpolarea orientarii folosind cuaternioni (1)

- Animatorul defineste o secventa de cadre cheie; pentru fiecare cadru :
  - Orientarea obiectului rigid/orientarile elementelor rigide ale structurii articulate sunt exprimate prin matrici de rotatie in jurul axelor principale (unghiuri Euler) ale sist. de coordonate locale.
- Matricile de rotatie corespunzatoare unui cadru sunt transformate (de sistemul de animatie) in cuaternioni (transformarea matrice de rotatie în cuaternion si invers →"Quaternions – optional")
- **Cuaternionii corespunzatori cadrelor cheie se numesc cuaternioni cheie.**
- La generarea cadrelor intermediare se interpoleaza intre cuaternionii cheie rezultand cuaternioni pentru fiecare cadru imagine.
- Cuaternionii corespunzatori unui cadru imagine sunt convertiti in matrici de rotatie in jurul axelor sistemului de coordonate locale al obiectului.
- Matricile de rotatie sunt folosite pentru calculul transformării varfurilor pentru cadrul imagine.

Interpolarea orientarii folosind cuaternioni (2)

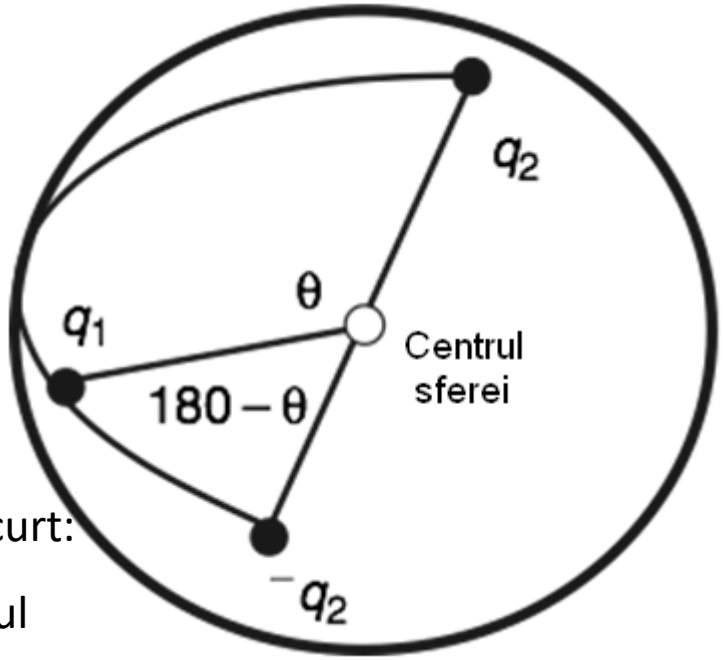
- O rotatie se reprezinta printr-un cuaternion unitate: → intregul grup de rotatii se reprezinta prin suprafata hiper-sferei unitate 4- dimensionale din spatiul cuaternion.
- Curbele de interpolare intre 2 orientari cheie se afla pe suprafata acestei sfere → trebuie efectuata **interpolare sferica liniara**.

- Intre oricare 2 puncte de pe suprafata sferei q1 si q2 exista 2 arce: q1 → q2 sau q1 → -q2

  q(s,v) si –q(-s, -v) reprezinta aceeasi orientare .

  → 2 posibilitati de interpolare intre 2 orientari:

  q1 → q2 sau q1 → -q2

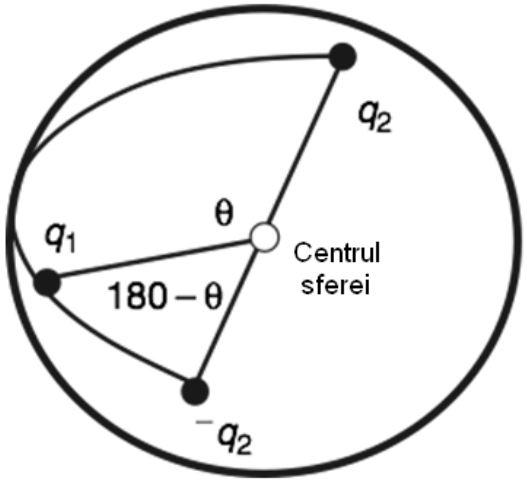


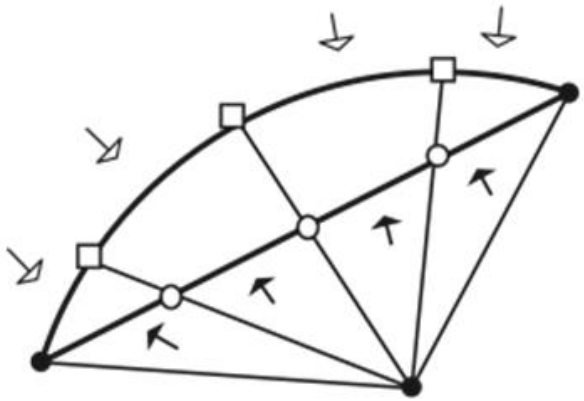
- Se prefera interpolarea de-a lungului arcului mai scurt: corespunde unghiului mai mic intre vectorii din spatiul cuaternion.

Interpolarea orientarii folosind cuaternioni (3)

- **Formula interpolarii sferice liniare** intre cuaternionii unitate q1 si q2, cu parametrul de interpolare u variind intre 0 si 1 (formula trebuie normalizata daca se doreste obtinerea unui cuaternion unitate):

slerp(q1, q2,u) = q1 * sin((1 – u)*Θ)/sin(Θ) + q2 * sin(u* Θ)/sin(Θ)





Interpolare liniara si interpolare sferica

- Cand Θ → 0 formula se reduce la interpolarea liniara: lerp(q1, q2, u) = (1-u) q1 + uq2
- La interpolarea sferica intre mai mult de 2 orientari cheie pot apare schimbari bruste de orientare (probleme de discontinuitate) la trecerea peste cadrele cheie. Problema poate fi rezolvata inlocuind interpolarea sferica liniara cu o interpolare cubica (ex. Bezier).

Quaternions

Prof. univ. dr. ing. Florica Moldoveanu

Cuaternioni (quaternions)

- Notiune matematica introdusa de matematicianul William Hamilton in 1843, in incercarea de a extinde planul numerelor complexe, a + ib ($i^2 = -1$) , pentru a defini un volum complex.
- A numit numarul

  q = a + bi + cj + dk ‘quaternion’

  cu proprietatea: $i^2 = j^2 = k^2 = -1$, ij = k; ji = -k
- Notatia prescurtata:

  $q = (s,v) = s + v_x i + v_y j + v_z k$, s este partea scalara (s – numar real) iar v partea vectoriala

  - (i,j,k) pot fi considerati ca versorii unui sistem de coord carteziene 3D iar **(v_x , v_y , v_z)** un vector raportat la acest sistem
- **In grafica se folosesc cuaternioni pentru a reprezenta orientarea prin axa si deplasament unghiular.**

Cuaternioni – proprietati (1)

- Este posibila:
  - -Interpolarea unica intre orientarea initiala si cea finala, reprezentate prin cuaternioni
  - -Concatenarea mai multor rotatii pentru a defini o transformare compusa prin cuaternioni

1) Rezultatul inmultirii a 2 cuaternioni este tot un cuaternion

   q1 = (s1, v1), q2 = (s2, v2)

   q1*q2 = (s1*s2 – v1•v2, s1*v2 + s2*v1 + v1xv2)

   • reprezinta produs scalar iar x reprezinta produs vectorial

2) Un punct in spatiu (vectorul din origine in punctul respectiv) se reprezinta prin:

   (0, v) – numit **cuaternion pur**

   (0, v1)*(0, v2) = (- v1•v2, v1 x v2) = (0, v1 x v2) daca v1•v2 = 0 (vectori ortogonali)

3) Conjugata unui cuaternion, q(s,v) este $\bar{q} = (s, -v)$

   Produsul dintre un cuaternion si conjugata sa defineste marimea cuaternionului:

$$q\bar{q} = s^2 + |v|^2 = |q|^2$$

Cuaternioni – proprietati (2)

4) **Cuaternionul unitate**: q = (s, v), unde $q\bar{q} = 1$

Atunci: q(cos(ϕ), sin (ϕ)N), cu |N| = 1, este un cuaternion unitate

5) Se defineste

$Rq(p) = q*p*q^{-1}$,unde

p = (0, r) este un cuaternion pur (r: vectorul de pozitie al unui punct)

q(s,v) este un cuaternion unitate

q^{-1} este inversul cuaternionului q: $q^{-1} = (1/|q|^2)\ (s, -v) = \bar{q}$ pentru cuaternionul unitate

$Rq(p) = (0, (s^2 - v \bullet v)*r + 2*v*(v \bullet r) + 2*s*vxr) = (0, (\cos(\phi)^2 - \sin(\phi)^2)*r +$

$2*\sin(\phi)^2*N*(N\bullet r) + 2*\cos(\phi) *\sin(\phi)*Nx\ r)$

$= \mathbf{(0, \cos(2\phi)*r + (1-\cos(2\phi))*N*(N\bullet r) + \sin(2\phi)*N\ x\ r)}$

Partea vectoriala a cuaternionului este similara cu expresia vectorului r rotit in jurul axei N (deplasamentul unghiular), cu diferenta ca unghiul este 2ϕ in loc de ϕ.

Reprezentarea rotatiilor prin cuaternioni(1)

CONCLUZIE:

- Un punct P avand vectorul de pozitie r in sistemul de coordonate local se reprezinta prin:
  $p = (0, r)$
- Deplasamentul unghiular cu ϕ, $R(\phi, N)$, se reprezinta prin cuaternionul unitate
  $q = (\cos(\phi/2), \sin(\phi/2)*N)$
- Rotatia punctului P in jurul axei N cu unghiul ϕ se exprima prin operatia $Rq(p)$

Avantajele utilizarii cuaternionilor pentru reprezentarea orientarii:

- Compunerea rotatiilor se obtine prin inmultirea cuaternionilor
  Inmultind 2 cuaternioni de marime 1 se obtine un cuaternion de marime 1:
  → 2 rotatii succesive in jurul aceleiasi axe, q, p, se reprezinta printr-o unica rotatie:
  $Rq(Rv(p)) = ((qv)*p*(qv)^{-1}) = Rqv(p)$

- **Folosind unghiuri Euler, o orientare se poate reprezenta prin rotatii diferite iar secventa de cadre obtinute prin interpolare intre cadre cheie nu este unica**:
  - o rotatie cu π in jurul axei X, $Rx(\pi)$, este echivalenta cu o rotatie compusa: $R = Ry(\pi)*Rz(\pi)$
  - la interpolare, rezulta secvente diferite:
    a) $R(0,0,0)$,……, $R(\pi t, 0,0)$,……, $R(\pi, 0, 0)$ $0 <= t <= 1$
    b) $R(0,0,0)$,……, $R(0, \pi t, \pi t)$,……., $R(0, \pi, \pi)$

Reprezentarea rotatiilor prin cuaternioni(2)

- **Folosind cuaternioni:**
  - Rx(π) se reprezinta prin: (cos(π /2), sin (π/2)[1,0,0]) = **(0, [1, 0, 0])**
  - Ry(π): **(0, [0,1,0])**
  - Rz(π): **(0, [0,0,1])**
  - Ry(π)*Rz(π) = (0, [0,1,0]x[0,0,1]) = **(0, [1,0,0])**

  - Concluzie: **cele 2 rotatii echivalente se reprezinta prin acelasi cuaternion→ interpolare unica intre 2 orientari reprezentate prin cuaternioni.**

Dezavantajul cuaternionilor in specificarea animatiei:

➢ **Reprezinta numai rotatie, nu si translatia**

- Transformarea varfurilor obiectului animat intre doua pozitii (cadre) cheie trebuie sa includa si translatie → orientarea reprezentata printr-un cuaternion , q(ϕ, N), care reprezinta deplasamentul unghiular, trebuie transformata in matrice de rotatie in jurul axelor sistemului de coordonate local al obiectului, pentru a putea fi compusa cu translatia.

Interpolarea orientarii folosind cuaternioni (1)

- Animatorul defineste o secventa de cadre cheie; pentru fiecare cadru :
  - Orientarea obiectului rigid/orientarile elementelor rigide ale structurii articulate sunt exprimate prin matrici de rotatie in jurul axelor principale (unghiuri Euler)
- Matricile de rotatie corespunzatoare unui cadru sunt transformate de sistemul de animatie in cuaternioni.
- Cuaternionii corespunzatori cadrelor cheie se numesc **cuaternioni cheie.**
- Se interpoleaza intre cuaternionii cheie rezultand cuaternioni intermediari pentru fiecare cadru imagine.
- Cuaternionii corespunzatori unui cadru imagine sunt convertiti in matrici de rotatie in jurul axelor sistemului de coordonate local al obiectului.
- Matricile de rotatie sunt folosite pentru calculul transformării varfurilor pentru cadrul imagine.

Interpolarea orientarii folosind cuaternioni(2)

- Utilizarea cuaternionilor pentru reprezentarea orientarilor presupune transformarea din matrice de rotatie in cuaternion si invers.

Conversia cuaternion → matrice de rotatie

Fie orientarea definita prin cuaternionul unitate:

$q = (\cos(\phi/2), \sin(\phi/2)*N) = (W, (X, Y, Z))$

unde X,Y,Z sunt componentele vectorului N (axa de rotatie)

si rotatia asupra unui vector de pozitie (punct) p definita prin:

$Rq(p) = q(p)q^{-1}$

Rotatia echivalenta a vectorului se obtine cu urmatoarea matrice de rotatie generala:

$$Rq = \begin{bmatrix} 1-2YY-2ZZ & 2XY-2WZ & 2XZ+2WY & 0 \\ 2XY+2WZ & 1-2XX-2ZZ & 2YZ-2WX & 0 \\ 2XZ-2WY & 2YZ+2WX & 1-2XX-2YY & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Conversia cuaternion→matrice de rotatie inseamna calculul acestei matrici.

Interpolarea orientarii folosind cuaternioni(3)

Conversia matrice de rotatie → cuaternion

- Presupune conversia unei matrici de rotatie generale, in formatul anterior, Rq

  → trebuie determinati W, X,Y,Z

$$R = \begin{bmatrix} M00 & M01 & M02 & 0 \\ M10 & M11 & M12 & 0 \\ M20 & M21 & M22 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Urma matriciei R este:

  urma (R) = M00 + M11 + M22 + M33

- Urma matricei Rq este = $4 - 4(X^2 + Y^2 + Z^2)$

- Cuaternionul echivalent este unitate, deci:

  $X^2 + Y^2 + Z^2 + W^2 = 1$

- Rezulta: urma(Rq) $= 4 - 4(1 - W^2) = 4W^2$ = urma(R) → **$W = ½(\text{urma}(R))^{0.5}$**

- W = cos(ϕ/2) → ϕ = 2*arccos(w)

Interpolarea orientarii folosind cuaternioni(4)

- Ceilalti termeni, (X, Y, Z) se obtin combinand elementele diagonal opuse ale matricei R:

  Mij - Mji , 0<= i,j<=2

  Rezulta:

  X = (M21 – M12)/4W Y = (M02 – M20)/4W Z = (M10 – M01)/4W

  X, Y, Z reprezinta componentele vectorului axa de rotatie scalati cu sin(ϕ/2).

  Pentru W=0 → X, Y, Z sunt nedefiniti.

  w=0 → cos(ϕ/2) = 0 → ϕ = ∏ sau ϕ = -∏

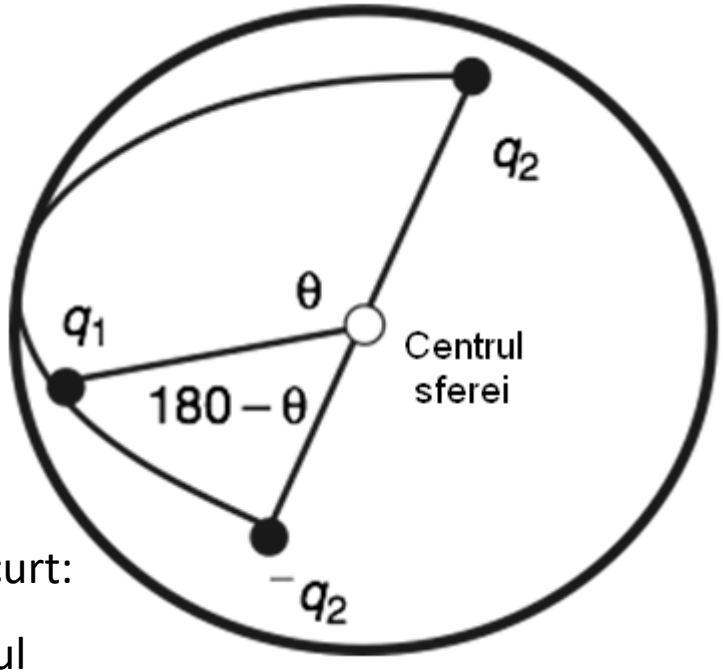
  X = sin(ϕ/2)*Nx

  Y = sin(ϕ/2)*Ny → ϕ = ∏ → X = Nx, Y = Ny, Z = Nz unde N = (Nx, Ny, Nz) este axa de rotatie

  Z = sin(ϕ/2)*Nz

Interpolarea in spatiul cuaternion(1)

- O rotatie se reprezinta printr-un cuaternion unitate: → intregul grup de rotatii se reprezinta prin suprafata hiper-sferei unitate 4- dimensionale din spatiul cuaternion.
- Curbele de interpolare intre 2 orientari cheie se afla pe suprafata acestei sfere → trebuie efectuata interpolare sferica liniara.



- Intre oricare 2 puncte de pe suprafata sferei exista 2 arce:

  q(s,v) si –q(-s, -v) reprezinta aceeasi orientare .

  → 2 posibilitati de interpolare intre 2 orientari:

  q1 → q2 sau q1 → -q2

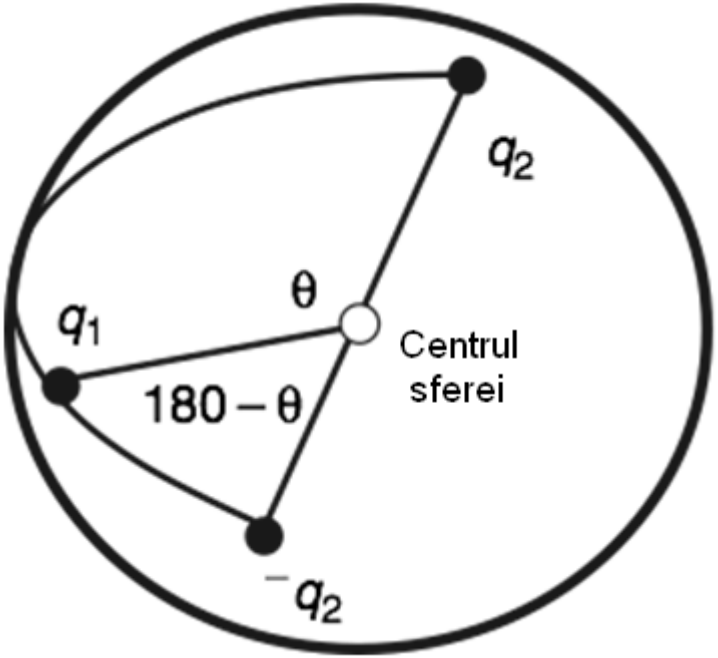
- Se prefera interpolarea de-a lungului arcului mai scurt: corespunde unghiului mai mic intre vectorii din spatiul cuaternion.

Interpolarea in spatiul cuaternion(2)

- Determinara arcului mai scurt se face calculand produsul scalar al cuaternionilor pentru a obtine cosinusul unghiului:

$$\cos(\theta) = q1 \bullet q2 = s1*s2 + v1 \bullet v2$$

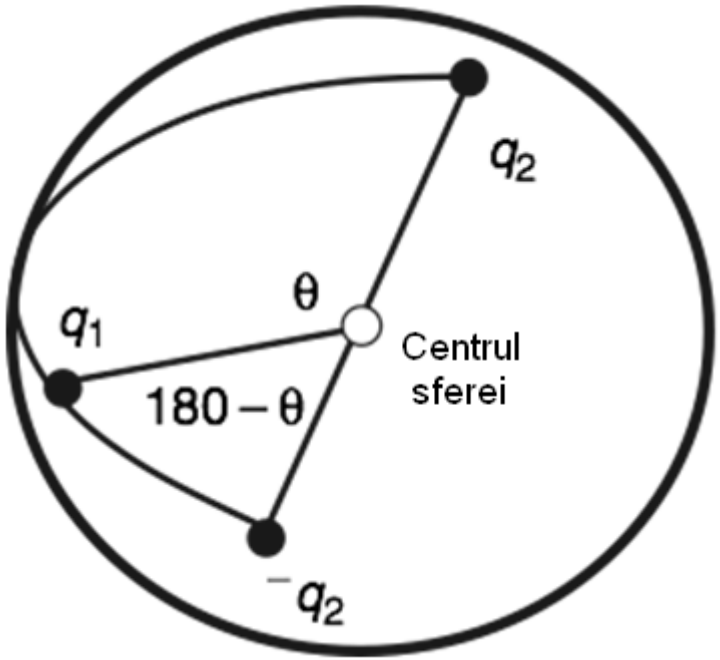
In acest exemplu, se va alege interpolarea de la q1 la –q2.

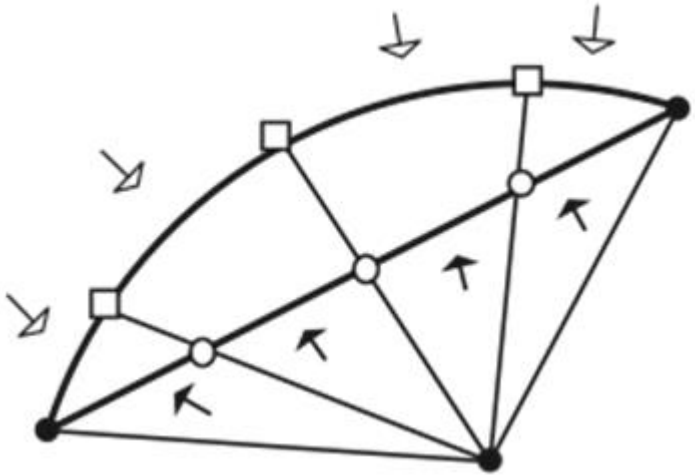


- Formula interpolarii sferice liniare intre cuaternionii unitate q1 si q2, cu parametrul u variind intre 0 si 1 (formula trebuie normalizata daca se doreste obtinerea unui cuaternion unitate):

slerp(q1, q2,u) = q1 * sin((1 – u)*Θ)/sin(Θ) + q2 * sin(u* Θ)/sin(Θ)

Interpolarea in spatiul cuaternion(3)





Interpolare liniara si interpolare sferica

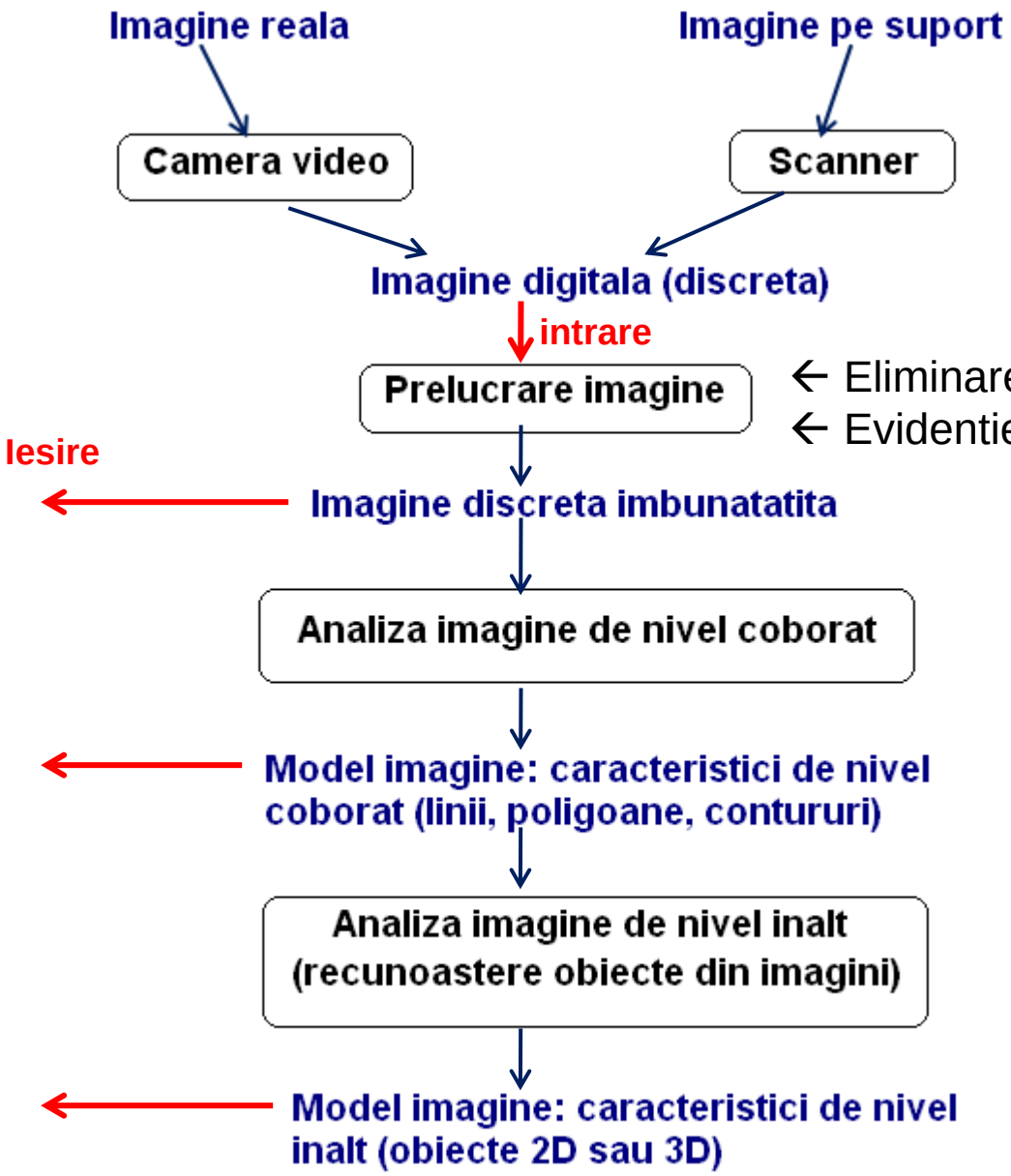
- Cand Θ → 0 formula se reduce la interpolarea liniara: lerp(q1, q2, u) = (1-u) q1 + uq2
- La interpolarea sferica intre mai mult de 2 orientari cheie pot apare schimbari bruste de orientare (probleme de discontinuitate)la trecerea peste cadrele cheie. Problema poate fi rezolvata inlocuind interpolarea sferica liniara cu o interpolare cubica (ex. Bezier).

Sisteme de prelucrare si analiza a imaginilor

Prof. univ. dr. ing. Florica Moldoveanu

Curs *Sisteme de Prelucrare Grafică* – UPB, Automatică şi Calculatoare
2021-2022

Prelucrarea si analiza imaginilor



←Sistem de formare a imaginii (sistem de achiziţie)

← Eliminare defecte (restaurare)
← Evidentiere (imbunatatire) caracteristici

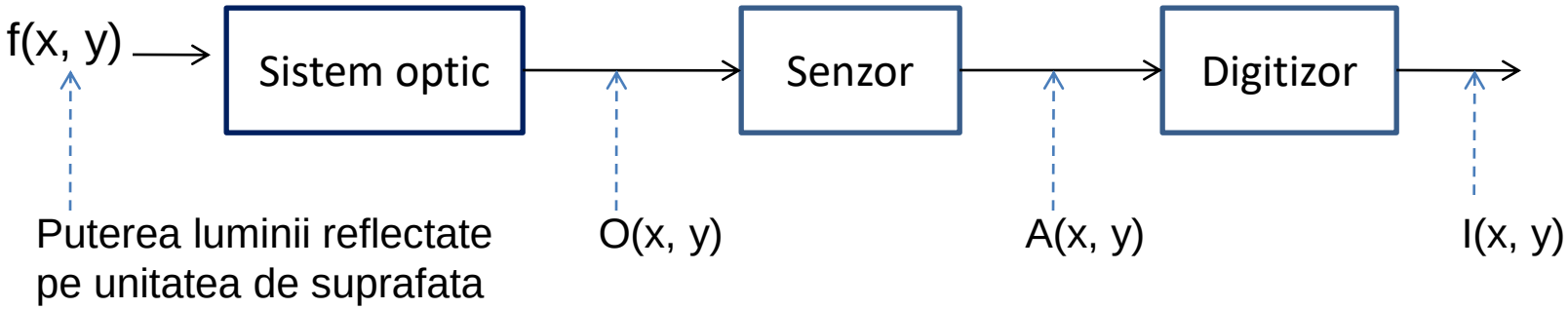
Sistem cu viziune (Computer Vision)

- prelucrare + analiza
- **scopul**: extragerea informaţiei conţinute in imagini
- **denumirea**: analogia cu sistemul vizual uman

Formarea imaginilor digitale(1)

Sistemul de formare a imaginilor digitale este compus din:

- **Sistem optic:** transforma energia luminoasa intr-un semnal optic 2D
- **Senzor:** semnalul optic → semnal electric analogic 2D
- **Digitizor video**: semnalul electric analogic → semnal electric digital 2D (imagine discreta)



Sistemul optic este modelat matematic printr-o convolutie 2D:

$$O(x, y) = h(x, y) * f(x, y) = \int\int h(\alpha, \beta)\, f(x - \alpha, y - \beta)\, d\alpha\, d\beta$$

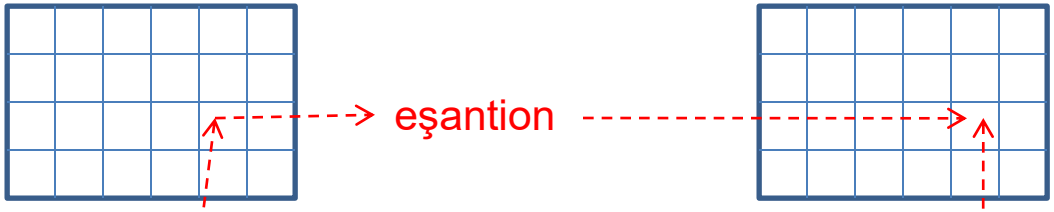
- Este un sistem de filtrare care elimina frecventele inalte continute in semnalul de intrare, care reprezinta informatie falsa.

Formarea imaginilor digitale(2)

Digitizarea: semnal electric analogic 2D → semnal electric digital 2D

- Eşantionarea
- Cuantizarea

Eşantionarea



P: planul imaginii analogice

Planul de afisare discret

Celula de digitizare:
subset compact si convex al lui P,
pe care se calculeaza valoarea unui esantion al imaginii analogice

Celula de afisare

Un esantion = element de imagine (picture element, pel, pixel)

- Prin afisarea eşantioanelor in celulele de afisare se reconstituie imaginea analogica.

Formarea imaginilor digitale(3)

Cuantizarea

- Valorile eşantioanelor (numere reale) → numere intregi (reprezentarea în memoria video).
- Daca un eşantion se reprezinta pe b biti

  → imaginea va avea 2^b niveluri de intensitate
- Eşantion: valoare reala [0,1]

  → pasul de cuantizare pentru reprezentare pe b biti este: $q= 1/(2^b)$

Rezolutia spaţiala a imaginii discrete: densitatea grilei de esantionare.

Rezolutia nivelului de gri: precizia cuantizarii (pasul de cuantizare) → nr. de biţi pe care se reprezinta un esantion.

Digitizarea imaginilor color

- este efectuata separat pe fiecare canal (3 imagini analogice): red, green, blue
- rezulta 3 imagini digitale

Restaurarea imaginilor

Restaurarea unei imagini (Image restoration) :

reconstruirea imaginii fără defectele introduse de sistemul de achizitie, cunoscand natura defectelor (defecte produse de lentilele optice, de senzor, focalizare gresita, miscarea relativa intre obiect si camera, s.a).

Operatii tipice:

- Corectia distorsiunilor geometrice
- Corectia defectelor produse de defocalizarea camerei
- Corectia defectelor produse de miscarea camerei sau a obiectelor din imagine (motion blur)
- Reducerea zgomotului din imagine

  (zgomot = pixel in imaginea discreta, care nu are corespondent in imaginea reala)

Imbunatatirea imaginilor

Imbunatatirea unei imagini (Image enhancement):

cresterea calitatii imaginii in scopul interpretarii sale vizuale sau a extragerii caracteristicilor.

Operatii tipice:

- Conversia nivelelor de gri (intensitate)
- Imbunatatirea contrastului
- Conversia culorilor
- Netezirea zgomotului
- Evidentierea caracteristicilor (de ex. contururi)

Operatiile de restaurare si imbunatatire a imaginilor = transformari de imagine:

- In domeniul spatial, (x,y)
- In domeniul frecventei (imaginea este un semnal electric 2D), asupra transformatei Fourier a imaginii

Restaurarea si Imbunatatirea imaginilor in domeniul spatial -1

Prof. univ. dr. ing. Florica Moldoveanu

Curs *Sisteme de Prelucrare Grafică* – UPB, Automatică şi Calculatoare
2021-2022

Conversia nivelelor de gri(1)

❖ Operatie de îmbunătățire a imaginilor

O imagine discreta:

este o functie

f : [0<= x <=N1-1, 0<= y = N2-1] → [0 - (L-1)], x,y numere intregi

unde: N1 si N2 reprezinta numarul de linii (N2) si de coloane (N1) ale imaginii

L este numarul de nivele de gri (intensitate) din imagine

f(x,y) = g, unde g este un nivel de gri (intensitate)

Conversia nivelelor de gri:

Functie de transformare T: [0 - (L-1)] → [0 - (L-1)]

daca

f(x,y) : imaginea de intrare

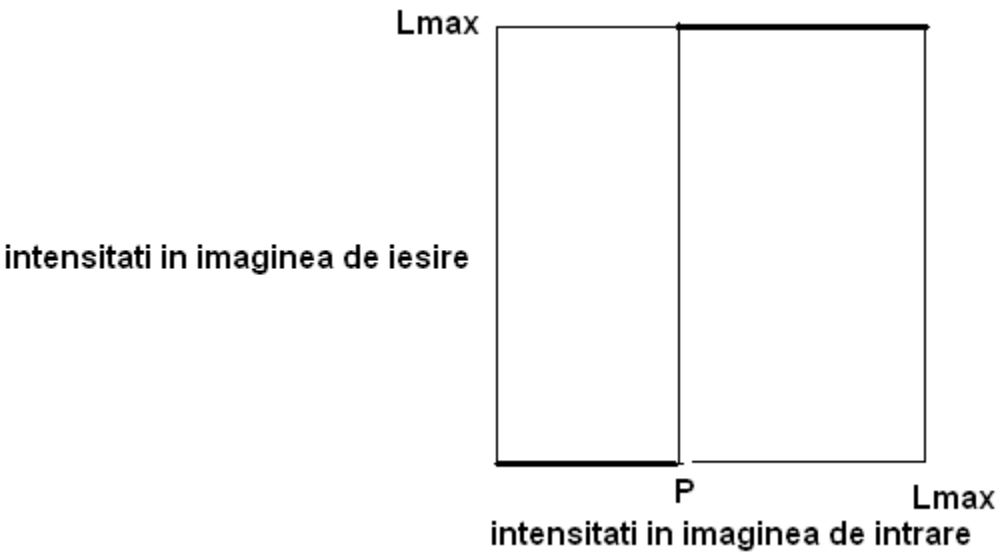
g(x,y): imaginea rezultata prin aplicarea transformarii T

atunci:

g(x,y) = T(f(x,y)), 0<= x <=N1-1, 0<= y = N2-1

Conversia nivelelor de gri(2)

Aplicarea unui prag (thresholding):



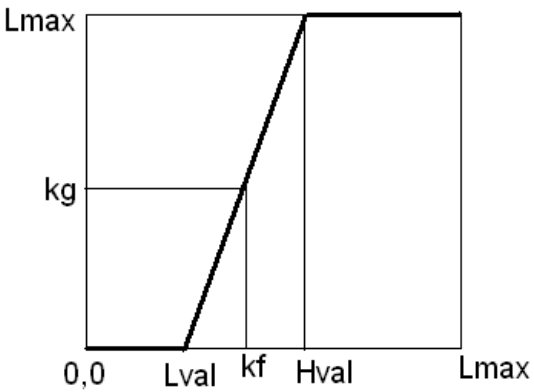
g(x,y) = 0 daca f(x,y) < P
Lmax daca f(x,y) > = P

unde P este „valoarea de prag“.

f(x,y) in L nivele de gri → g(x,y), imagine binara

Conversia nivelelor de gri(3)

Imbunatatirea contrastului (Contrast stretching) - Normalizarea



$$kg = T(kf) = \begin{cases} 0, & kf <= Lval \\ Lmax & kf >= Hval \\ s(Kf - Lval), & Lval < kf < Hval \end{cases}$$

unde $s = Lmax/(Hval - Lval)$

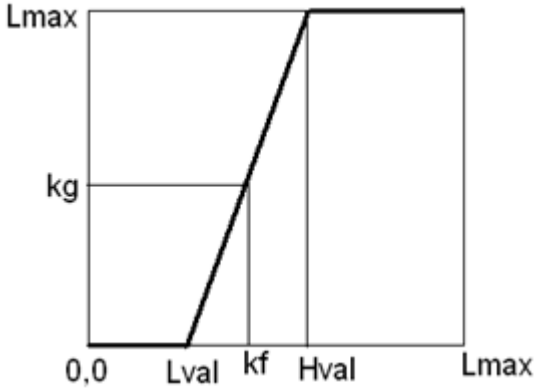
sau:

$$g(x,y) = \begin{cases} 0 & \text{daca } f(x,y) <= Lval \\ Lmax & \text{daca } f(x,y) >= Hval \\ s * (f(x,y) - Lval) & \text{daca } Lval < f(x,y) < Hval \end{cases}$$

Intensitatile din imaginea de intrare cuprinse intre Lval şi Hval (un domeniu ingust de intensitati) sunt scalate a.i. sa fie uniform distribuite in imaginea de iesire.

Conversia nivelelor de gri(4)

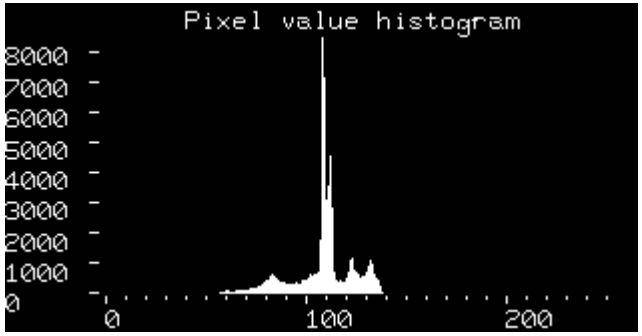
Imbunatatirea contrastului - normalizarea

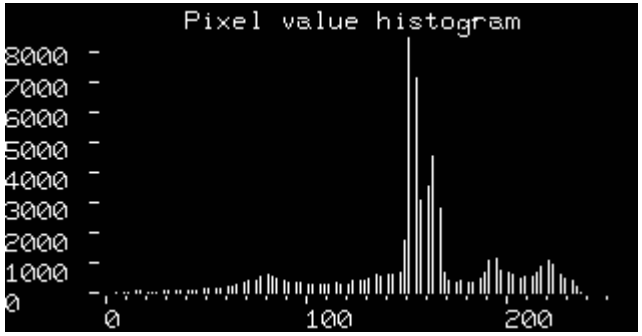


De regula:

- Lval este cea mai mica intensitate din imaginea de intrare
- Hval este cea mai mare intensitate din imaginea de intrare

Histograma imaginii inainte si dupa operatia de normalizare





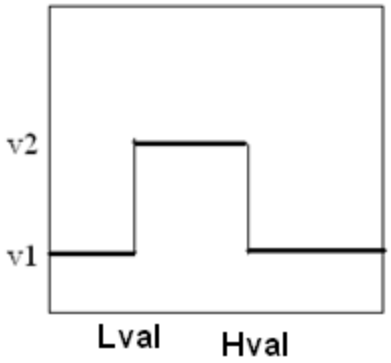
Lval=79
Hval = 136

https://homepages.inf.ed.ac.uk/rbf/HIPR2/stretch.htm

Conversia nivelelor de gri(5)

Evidenţierea caracteristicilor

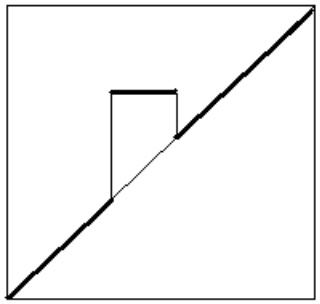
Se evidenţiaza un domeniu specific de niveluri de gri din imagine, reducand totodata nivelurile din afara domeniului la un nivel constant.



g(x, y) = v1 daca f(x,y) < =Lval sau f(x,y) >= Hval
v2 altfel

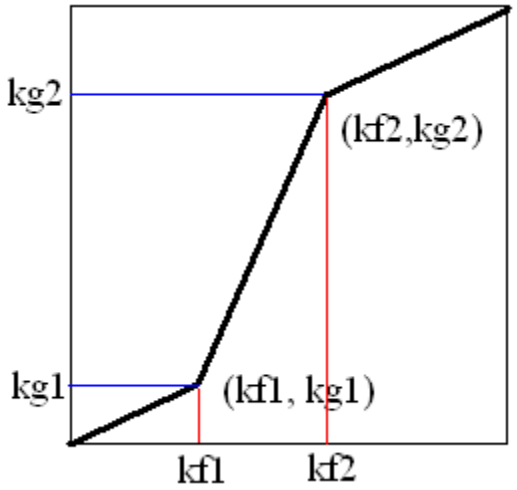
Intensitatile dintre Lval si Hval pot reprezenta: obiecte de interes din imagine (celule intr-o img medicala), contururi, etc.

Se evidentiaza un domeniu specific de niveluri de gri din imagine, lasand nemodificate nivelurile din afara domeniului.



g(x, y) = f(x,y) daca f(x,y) < =Lval sau f(x,y) >= Hval
v2 altfel

Conversia nivelelor de gri(6)



Cazul general

Functia de transformare: functie liniara definita pe portiuni

kg = kf * s1, s1 = kg1 / kf1 pentru 0<= kf <= kf1

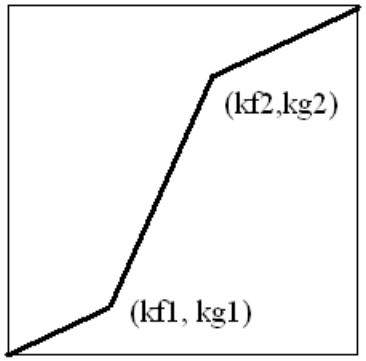
kg = kf * s2 + kg1 – kf1*s2, s2 = (kg2 – kg1)/ (kf2-kf1) pentru kf1 < kf <kf2

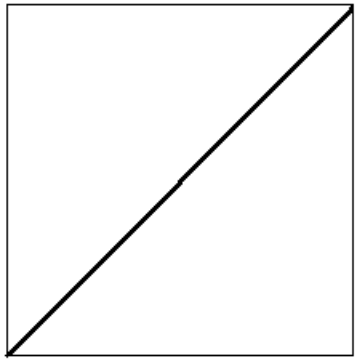
kg = kf * s3 + kg2 – kf2*s3, s3 = (Lmax – kg2) / (Lmax- kf2) pentru kf2 <= kf <=Lmax

kg1 < kf1 si kg2 > kf2 → mărire contrast: (kf2-kf1) < (kg2-kg1) – cazul din figura

kg1 > kf1 si kg2 < kf2 → reducere contrast : (kf2-kf1) > (kg2-kg1)

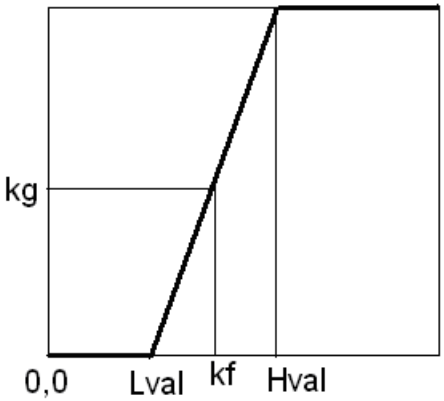
Conversia nivelelor de gri(7)





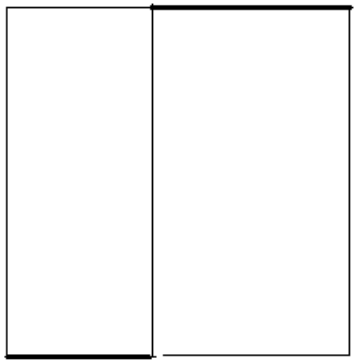
kf1=kg1, kf2=kg2

Nu se produc schimbari de intensitate



kf1=Lval, kg1 =0, kf2=Hval, kg2=Lmax

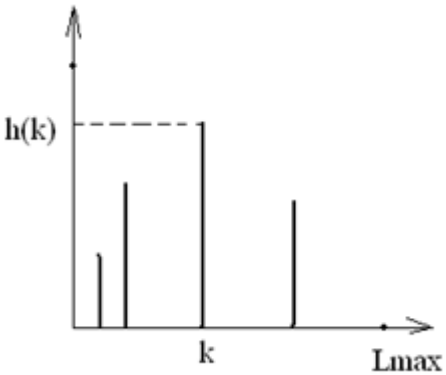
Contrast stretching
(imbunatatirea contrastului)



kf1=kf2, kg1=0, kg2=Lmax

Thresholding

Conversia nivelelor de gri folosind histograma imaginii



Histograma unei imagini: functie discreta,

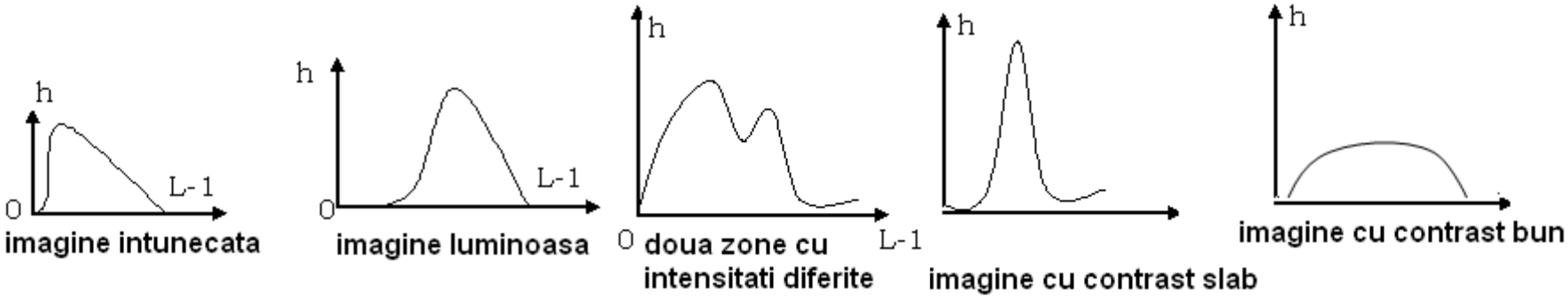
$h(k) = n_k$

n_k este numarul de pixeli din imagine, cu intensitatea k

Histograma normalizata:

$h(k) = n_k / n \quad \rightarrow \quad 0 <= h(k) <= 1$

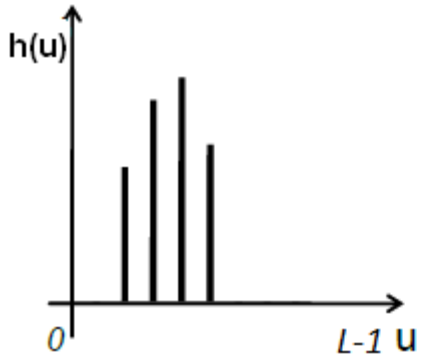
n este numarul total de pixeli din imagine



Egalizarea histogramei (1)

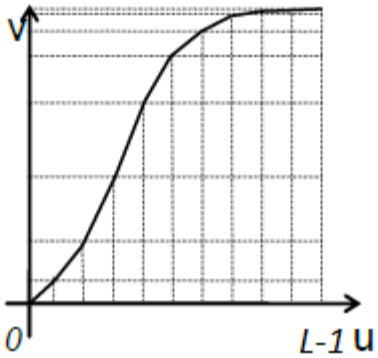
Imbunatatirea contrastului prin egalizarea histogramei

Histograma imaginii de intrare (I)



Inensitatile din imaginea de intrare

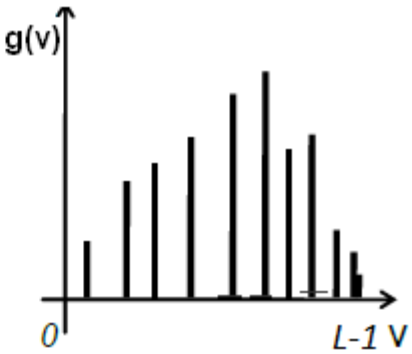
$h_k = nI_k/n$



Graficul transformarii intensitatilor din imaginea de intrare

$v = T(u)$

Histograma imaginii de iesire (E)



Intensitatile din imaginea de iesire

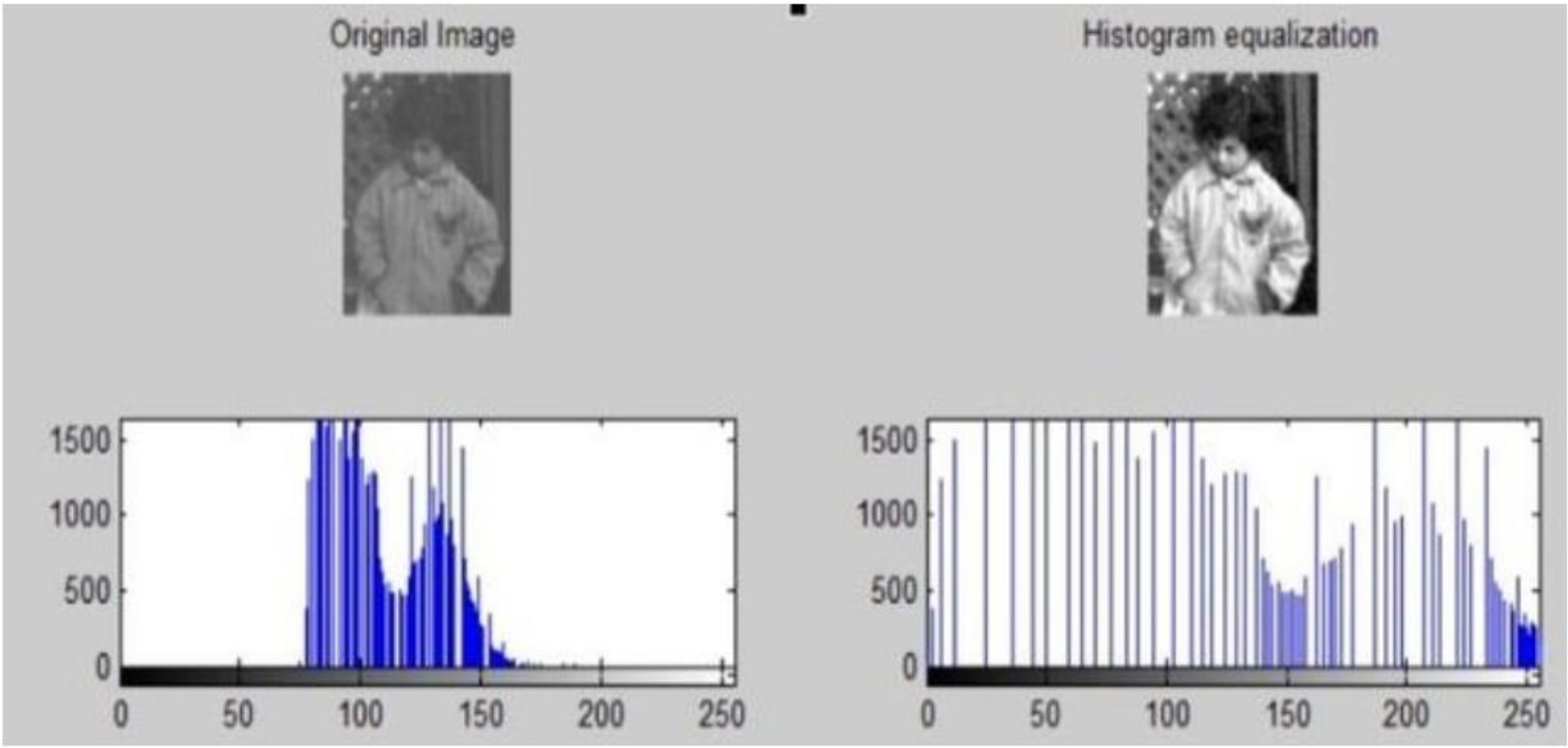
$g_k = nE_k/n = \sum_{j=0}^{j=k} \frac{nIj}{n} = \sum_{j=0}^{j=k} hj$

$$v_k = T(u_k) = (L-1)\sum_{j=0}^{j=k} \frac{nIj}{n} = (L-1)\sum_{j=0}^{j=k} hj$$

Egalizarea histogramei: egalizarea contributiei nivelurilor de intensitate din imagine.

- Transformarea T trebuie sa produca o imagine cu o histograma aplatisata.

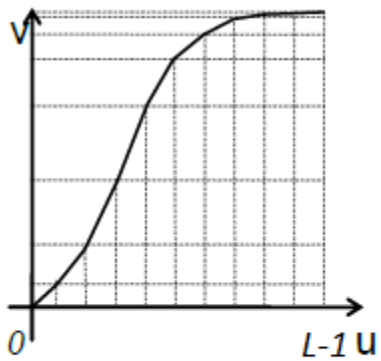
Egalizarea histogramei (2)



https://www.researchgate.net/publication/324108996_Histogram_equalization_and_noise_removal_process_for_enhancement_of_image/figures

Egalizarea histogramei (3)

- Considerand intensitatile din cele 2 imagini, u şi v, ca variabile aleatoare continue pe domeniul [0 - L-1]



b) graficul transformarii intensitatilor din imaginea initiala

v = T(u) = (L-1)* g_u

$$V = T(U) = (L-1)\int_0^U p_U(u)du,$$

p_U: functia de densitate a probabilitatii intensitatilor u

integrala: distributia cumulativa a intensitatilor din imaginea de intrare

- se aproximeaza discret prin ∑ p(u)

❖ Histograma egalizata este definita folosind functia de distributie cumulativa a intensitatilor din imaginea de intrare (histograma normalizata acumulata):

$$vk = (L-1)\sum_{j=0}^{k}\frac{n_j}{n} = (L-1)\sum_{j=0}^{k}h_j$$

Egalizarea histogramei (4)

Calculul histogramei normalizate

```
typedef unsigned char **imagine; // Intensitatea maxima: 255
 void histo(imagine a, int H, int W, int L, float *h)
// W si H: latimea si inaltimea imaginii; L: numarul de niveluri de gri din imagine
{unsigned long*lh; float n;
 * alocare lh ; // in lh se calculeaza histograma imaginii
 for (i=0; i<L; i++) lh[i]=0;
 for (i=0; i<H; i++)
      for (j=0; j<W; j++)
            lh[ (unsigned int) a[i][j]]++; // a[i][j] este intensitatea pixelului de pe linia i, coloana j
// se normalizeaza histograma
 n=(float) H*W;
 for (i=0; i<L; i++)
      h[i]=lh[i]/n;
 * dealocare lh;
}
```

Egalizarea histogramei (5)

Calculul imaginii imbunatatite pe baza histogramei egalizate

void EgalHisto (imagine In, int H, int W, int L, imagine Out)

histo(In, H, W, L, h); // Calculeaza histograma normalizata a imaginii de intrare, h

// Calculeaza histograma egalizata:

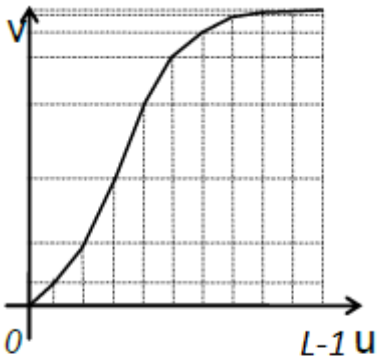
```
  g[0] = h[0];
  for ( i=1; i<L; i++ )
      g[i] = g[i-1] + h[i]
```

$$g_k = \sum_{j=0}^{k} h_j$$

// Calculeaza intensitatile din imaginea de iesire

```
for(i = 0; i<L; ++ )
     f[i] = g[i]* (L-1);
```

$$vk = (L-1) \sum_{j=0}^{k} \frac{n_j}{n} = (L-1) \sum_{j=0}^{k} h_j$$



Graficul transformarii intensitatilor din imaginea de intrare

v = T(u)

// Calculeaza imaginea de iesire

```
 for(i = 0; i<H; I++ )
     for(j = 0; j<W; j++ )
      Out[i][j] = ((unsigned char) f[(unsigned int)In[i][j]];
* dealocare f, h, g
}
```

Filtre spatiale

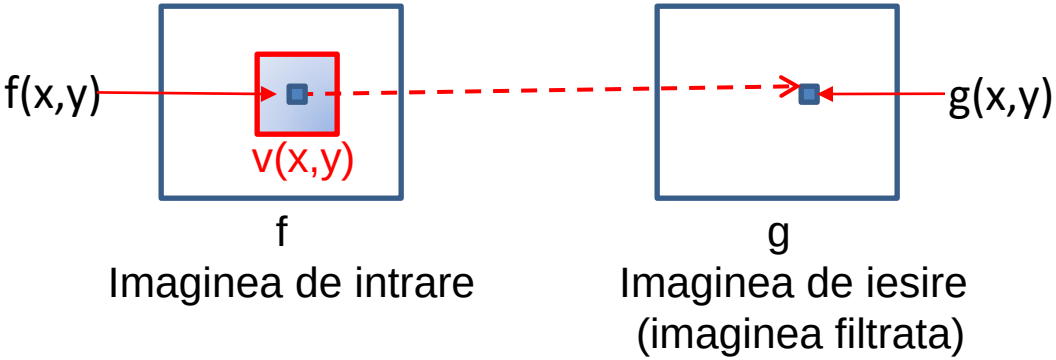
Prof. univ. dr. ing. Florica Moldoveanu

Curs *Sisteme de Prelucrare Grafică* – UPB, Automatică şi Calculatoare
2021-2022

Filtre spatiale(1)

Filtrarea spațială a imaginilor digitale

- **Scopul**: reducerea zgomotului din imagini sau evidentierea anumitor caracteristici, prin procesarea reprezentarii imaginii în domeniul spațial.
- Operatia se aplica local, la nivelul fiecarui pixel din imagine, inlocuind valoarea pixelului curent, f(x,y), cu o valoare, g(x,y), ce depinde de valorile pixelilor dintr-o vecinatate v(x,y) a pixelului (x,y).
- Se considera o vecinatate spatiala, in sistemul de coordonate asociat imaginii
- Se folosesc vecinatati mici, între 3x3 şi 11x11 pixeli.



Filtre spatiale(2)

***Filtre de mediere*: filtre liniare**

- g(x,y) este o medie ponderata a valorilor pixelilor din vecinatatea v(x,y);

- mai eficiente pentru imagini cu *zgomot uniform* sau *zgomot gaussian;*

***Filtrele ordonate (filtre rang), filtrul conservativ:* filtre neliniare**

- Filtrul rang: g(x,y) este una dintre valorile pixelilor din vecinatatea v(x,y), aleasa pe baza pozitiei sale in vectorul ordonat crescator al valorilor pixelilor din vecinatate (prima, ultima, la mijloc);
- mai eficiente pentru imaginile cu *zgomote sare-si-piper, zgomote exponential negative* si *zgomote Rayleigh.*

***Filtre adaptive*:**

- îsi schimba comportamentul in functie de caracteristicile nivelelor de gri din vecinatatea v(x,y);
- des folosite in multe aplicatii practice;

Filtre de mediere

- Valoarea unui pixel(x,y) din imaginea filtrata este o **medie ponderata** a valorilor pixelilor dintr-o vecinatate a pixelului, numită **fereastra de filtrare**.
- Ponderile sunt definite într-o matrice de forma vecinatatii, numita ***masca de convolutie*** sau **filtru** sau **kernel**.
- Aplicarea filtrului peste intreaga imagine este numita adesea ***convolutia imaginii*** cu ***masca*** de convolutie.

$$g(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} w(i,j) * f(x+i, y+j)$$

f(x,y): imaginea de intrare
g(x,y): imaginea filtrata
w: masca de convolutie

Exemplu: pentru o fereastra de filtrare de 3x3 pixeli, a = 1, b = 1

Filtrul trece-jos (Low-pass filter)

Numit si filtru de netezire - toate ponderile w(i,j) sunt pozitive
- suma ponderilor = 1 → aplicarea filtrului pe o zona de intensitate constanta nu produce schimbari

Exemplu:

w(0,0) = ¼

w(1,0) = w(-1, 0) = w(0,1) = w(0,-1) = 1/8

w(-1,-1) = w(-1, 1) = w(1,-1) = w(1,1) = 1/16

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

Netezirea tranzitiilor bruste de intensitate

- Un filtru trece-jos (FTJ) atenueaza componentele de inalta frecventa din imagine (tranzitii bruste de intensitate intre pixeli vecini), care pot reprezenta zgomote.
- Imaginea rezultata din aplicarea unui FTJ este mai incetostata (neclara) decat imaginea originala.
- Prin atenuarea tranzitiilor bruste de intensitate se pot pierde detalii din imagine.

Filtrul trece-jos - exemplu de aplicare

1	2	3	4	5	6	7	8	9	10	11
0	0	0	50	50	50	50	0	0	0	0
0	0	0	50	50	50	50	0	0	0	0
0	0	0	50	50	50	50	0	0	0	50
0	0	0	50	50	50	50	0	0	0	0

Imaginea de intrare

Zgomot "sare si piper"

frontiere

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

1	2	3	4	5	6	7	8	9	10	11
0	0	0	50	50	50	50	0	0	0	0
0	0	12,5	37,5	50	50	37,5	12,5	0	3,125	
0	0	12,5	37,5	50	50	37,5	12,5	0	6,25	
0	0	0	50	50	50	50	0	0	0	0

Imaginea filtrată

Zgomotul a fost redus si împrastiat

Filtrul medie (1)

Caz particular de FTJ

- g(x,y) este media aritmetica a pixelilor din vecinatate

Exemplu:

$$g(x,y) = 1/9 \sum_{i=-1}^{1} \sum_{j=-1}^{1} f(x+i, y+j)$$

Filtrul medie (2)

Implementarea filtrului medie

Functia f_net aplica filtrul medie pe un dreptunghi din imaginea a, reprezentat prin colturile sale (M1, N1), (M2, N2), rezultatul fiind memorat in imaginea b.

Fereastra de filtrare are w x h pixeli.

```
void f_net (imagine a, imagine b, int w, int h, int N1, int M1, int N2, int M2)
{  float wh, s;
   wh= (float)(w*h); // numarul de pixeli din fereastra de filtrare
   w2 = w/2; h2 = h/2;
   for ( l = M1 + h2;  l < M2 – h2;  l++)
      for ( k = N1 + w2;  k < N2 – w2;  k++)
       { s=0;
         for ( i = -h2;  i <= h2;  i++)
               for ( j = -w2;  j < w2;  j++)
                              s + = a[l+i][k+j];
        b[l][k] = (unsigned char) (s/wh);
      }
}
```

Filtrul nu se aplica pe marginile zonei filtrate.
De ex: h=w=3 →h/2=w/2=1
→ filtrul nu se aplica pe liniile M1, M2 şi coloanele N1, N2

Filtrul trece-sus (High-pass filter)

- Un filtru trece sus (FTS) accentueaza schimbarile de intensitate.
- Masca filtrului este proiectata a.î. sa mărească intensitatea pixelului din centru relativ la ceilalti pixeli din vecinatate. De regula singura valoare pozitiva din masca este in centru.

Exemple:

W(0,0) = 9

w(i,j) = -1, pentru toate celelalte elemente

1	-2	1
-2	5	-2
1	-2	1

W(0,0) = 5

W(-1,0) = w(1,0) = w(0,-1) = w(0,1) = -1

W(1,1) = w(1,-1) = w(-1,-1) = w(-1,1) = 0

0	-1	0
-1	5	-1
0	-1	0

Suma ponderilor = 1 → filtrul lasă neschimbate zonele de intensitate constanta.

❖ FTS este folosit pentru imbunatatirea contrastului dintre zone adiacente între care exista o diferenta mica de intensitate, în scopul evidentierii frontierelor şi a detaliilor din imagini.

Filtrul trece-sus - exemplu de aplicare

1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	50	50	50	50	0	0	0	0	0
0	0	0	50	50	50	50	0	0	0	0	0
0	0	0	50	50	50	50	0	0	0	50	0
0	0	0	50	50	50	50	0	0	0	0	0

Imaginea de intrare

0	-1	0
-1	5	-1
0	-1	0

frontiere

1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	50	50	50	50	0	0	0	0	
0	0	-50	100	50	50	100	-50	0	0	-50	
0	0	-50	100	50	50	100	-50	0	-50	250	
0	0	0	50	50	50	50	0	0	0	0	

Valorile obtinute prin aplicarea filtrului sunt limitate la intervalul [0, val de intensitate maxima]

1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	50	50	50	50	0	0	0	0	
0	0	0	100	50	50	100	0	0	0	0	
0	0	0	100	50	50	100	0	0	0	250	
0	0	0	50	50	50	50	0	0	0	0	

Imaginea filtrata

Evidentierea frontierelor (Unsharp filter)

- FTS amplifică zgomotele din imagine si detaliile mici (nesemnificative).

  Solutia: Unsharp filter (“unsharp” – neascutit)

- Fie: g(x,y) = f(x,y) – fnet(x,y)

  unde f(x,y) este imaginea de intrare

  fnet(x,y) este imaginea de intrare netezita prin aplicarea unui filtru trece-jos

g(x,y) are valori diferite de zero in pixelii de frontiera din imaginea de intrare

- **Filtrul “Unsharp” este definit astfel:**

  funsharp(x,y) = f(x,y) + k*g(x,y)

  k este un factor de scara cuprins intre 0.2 – 0.7;

  valorile mari ale lui k amplifica frontierele

funsharp este imaginea de intrare cu frontierele evidentiate.

Unsharp filter - exemplu de aplicare

1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	50	50	50	50	0	0	0	0	0
0	0	0	50	50	50	50	0	0	0	0	0
0	0	0	50	50	50	50	0	0	0	50	0
0	0	0	50	50	50	50	0	0	0	0	0

Imaginea de intrare: f(x,y)

frontiere

1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	50	50	50	50	0	0	0	0	0
0	0	12,5	37,5	50	50	37,5	12,5	0	3,125	6,25	0
0	0	12,5	37,5	50	50	37,5	12,5	0	6,25	12,5	0
0	0	0	50	50	50	50	0	0	0	0	0

Imaginea netezita: fnet(x,y)

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	50	50	50	50	0	0	0	0	0
0	0	0	12,5	0	0	12,5	0	0	0	0	0
0	0	0	12,5	0	0	12,5	0	0	0	37,5	0
0	0	0	50	50	50	50	0	0	0	0	0

g(x,y) = f(x,y) - fnet(x,y)

1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	50	50	50	50	0	0	0	0	0
0	0	0	56,25	50	50	56,25	0	0	0	0	0
0	0	0	56,25	50	50	56,25	0	0	0	68,75	0
0	0	0	50	50	50	50	0	0	0	0	0

funsharp(x,y) = f(x,y) + 0.5*g(x,y)

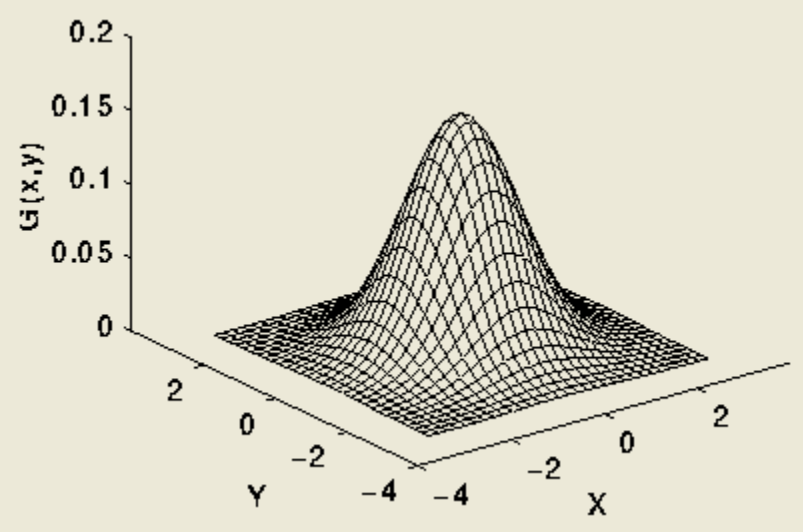
Filtrul Gaussian(1)

Functia de distributie Gaussiana 2D, cu media in (0,0) este:

$$g(x,y,\sigma) = (1/(2\pi\sigma^2))\exp(-(x^2+y^2)/(2\sigma^2))$$

σ = deviatia standard

- Functia se anuleaza la o distanta de aproximativ $3*\sigma$ faţă de medie.



Distributia Gausiana 2D cu m=(0,0) si σ = 1

Filtrul Gaussian(2)

- *Masca filtrului Gaussian* este o aproximare discreta a functiei de distributie Gaussiana 2D
- Se considera nula valoarea functiei la o distanta de 3σ fata de medie

$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

$\sigma = 1.0$

$\frac{1}{115}$

2	4	5	4	2
4	9	12	9	4
5	12	15	12	5
4	9	12	9	4
2	4	5	4	2

$\sigma = 1.4$

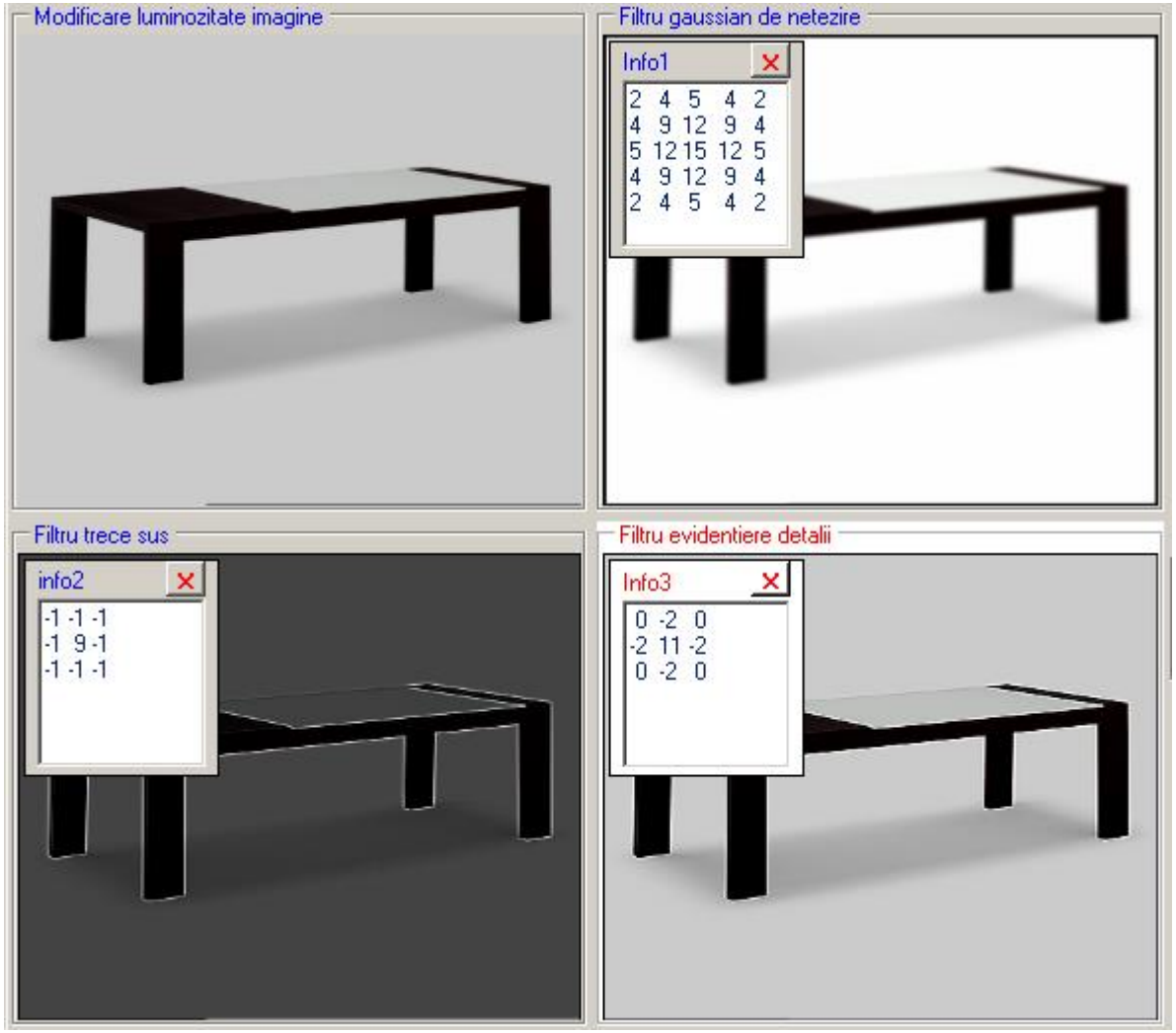
$$M(x,y) = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

masca de 3x3 care aproximeaza convolutia Gaussiana

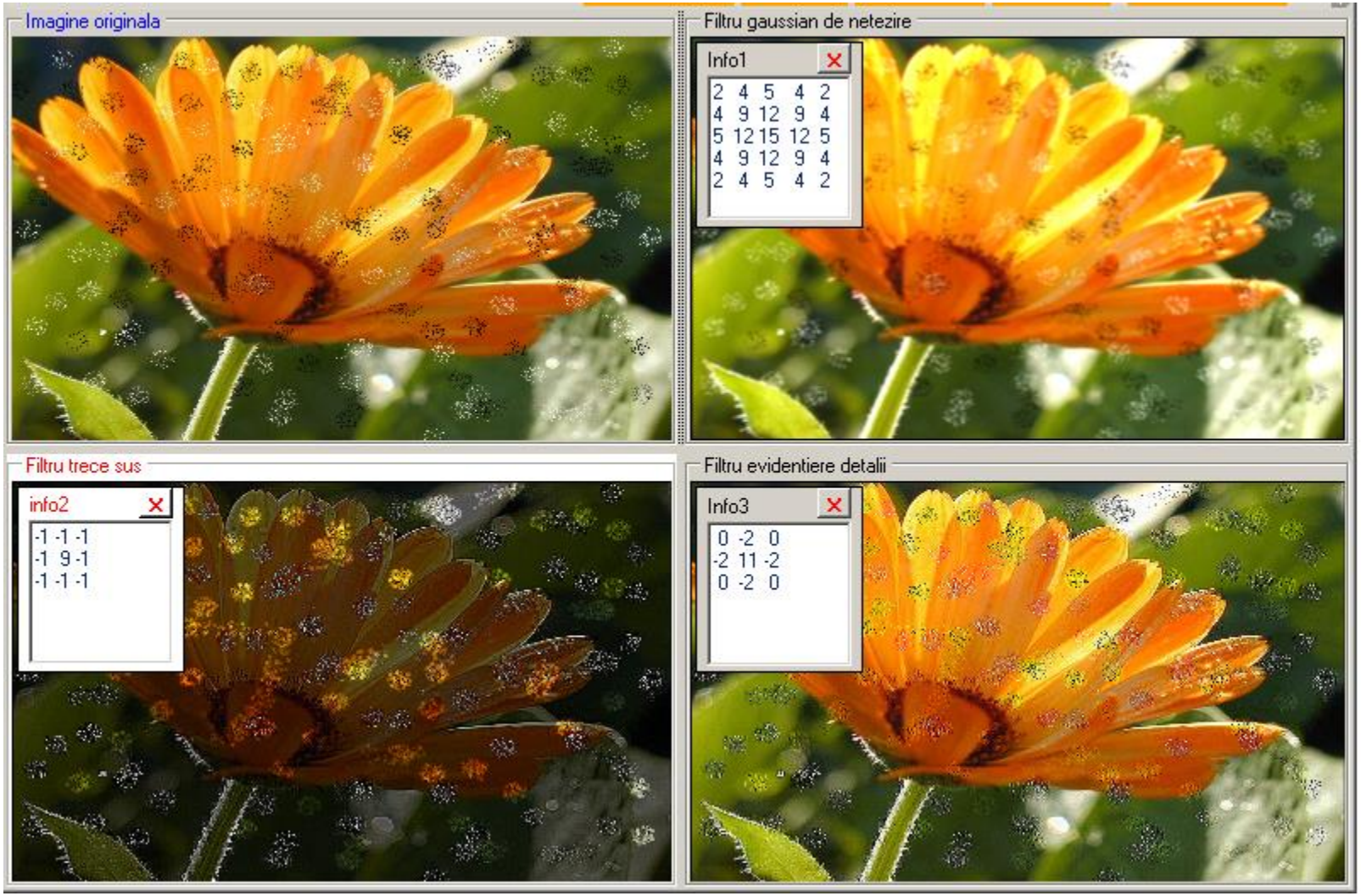
Filtrul Gaussian(3)

- Filtrul Gaussian este un filtru de netezire.
- Gradul de netezire este determinat de valoarea deviatiei standard (creste odata cu σ).
- σ determina dimensiunea mastii: se alege in functie de nivelul de zgomot din imagine.
- filtrul Gaussian produce o netezire mai fina decat filtrul medie
- conserva frontierele mai bine decat un filtru medie de aceeasi dimensiune
- Filtrul gaussian poate fi folosit pentru eliminarea *zgomotului gaussian* si a *zgomotului uniform.*

Filtre de mediere – exemple (1)



Filtre de mediere – exemple (2)



Filtre ordonate (1)

Doua tipuri de filtre spatiale:

- **liniare**: iesirea filtrului este o combinatie liniara a valorilor pixelilor din fereastra de filtrare, definita prin masca de convolutie: filtrele de mediere
- **neliniare**: iesirea filtrului este o combinatie ne-liniara a valorilor pixelilor din fereastra de filtrare; difera de la un tip de filtru la altul. Ex: filtrele ordonate, filtrul conservativ, s.a.

Filtre ordonate (filtre rang)

- Filtre neliniare
- Folosesc ordonarea intensitatilor pixelilor din fereastra de filtrare
- Reduc zgomotele (frecvente inalte) fara a degrada detaliile din imagine (fronturile)
- Nu produc nivele noi de intensitate in imaginea filtrata

P(x, y): pixelul curent

FP(x,y) = FP[0]<=FP [1]<= FP [2]<=….<=FP [n-1],

vectorul ordonat al intensitatilor din fereastra de filtrare pentru pixelul P

Filtre ordonate (2)

Filtrul rang: g(x,y) = Rang-j (FP(x,y))

- intensitatea pixelului (x,y) din imaginea filtrata este aceea din pozitia j a vectorului FP

j=0: **filtrul min** - g(x,y) = FP[0]

j=n-1: **filtrul max** – g(x,y) = FP[n-1]

j = n/2, n impar : filtrul median

100	110	120
130	125	124
150	250	110

Proprietatile filtrului median

1) Inlatura zgomotele fara a atenua punctele de frontiera (tranzitie brusca de intensitate):

FP= 100, 110, 110, 120, 124, 125, 130, 150,250

→ 250 poate fi un zgomot sau o valoare caracteristica din imagine

- iesirea filtrului median: valoarea pixelului din centrul ferestrei este inlocuita cu 124 (elementul din pozitia 9/2 a vectorului ordonat al intensitatilor)

- iesirea filtrului medie: valoarea pixelului din centrul ferestrei este inlocuita cu 1215/9 = 135, valoare de intensitate inexistenta in fereastra de filtrare

Filtul median – exemplu de aplicare

Imaginea de intrare

1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	50	50	50	50	0	0	0	0	0
0	0	0	50	50	50	50	0	0	0	0	0
0	0	0	50	50	50	50	0	0	0	50	0
0	0	0	50	50	50	50	0	0	0	0	0

frontiere

1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	50	50	50	50	0	0	0	0	0
0	0	0	50	50	50	50	0	0	0	0	0
0	0	0	50	50	50	50	0	0	0	50	0
0	0	0	50	50	50	50	0	0	0	0	0

frontiere

1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	50	50	50	50	0	0	0	0	
0	0	12,5	37,5	50	50	37,5	12,5	0	3,125		
0	0	12,5	37,5	50	50	37,5	12,5	0	6,25	12,5	
0	0	0	50	50	50	50	0	0	0	0	

Imaginea obtinuta prin aplicarea filtrului de netezire

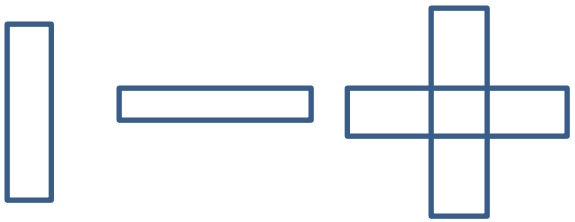
1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	50	50	50	50	0	0	0	0	
0	0	0	50	50	50	50	0	0	0	0	
0	0	0	50	50	50	50	0	0	0	0	
0	0	0	50	50	50	50	0	0	0	0	

Imaginea obtinuta prin aplicarea filtrului median cu o fereastra de 3x3 pixeli

Filtre ordonate (3)

Proprietatile filtrului median - cont



2) Forma ferestrei de filtrare influenteaza efectul filtrului

- se alege mica (pentru evitarea efortului de sortare)

- alegerea formei se poate baza pe cunoasterea caracteristicilor de zgomot din imagine; poate fi: patrat, dreptunghi (orizontal/vertical), cruce.

3) Reduce variatia intensitatilor din imagine, producand regiuni de intensitate constanta sau aproape constanta. Forma regiunilor depinde de geometria ferestrei de filtrare, aspect neplacut deoarece sunt introduse in imaginea filtrata regiuni care nu existau.

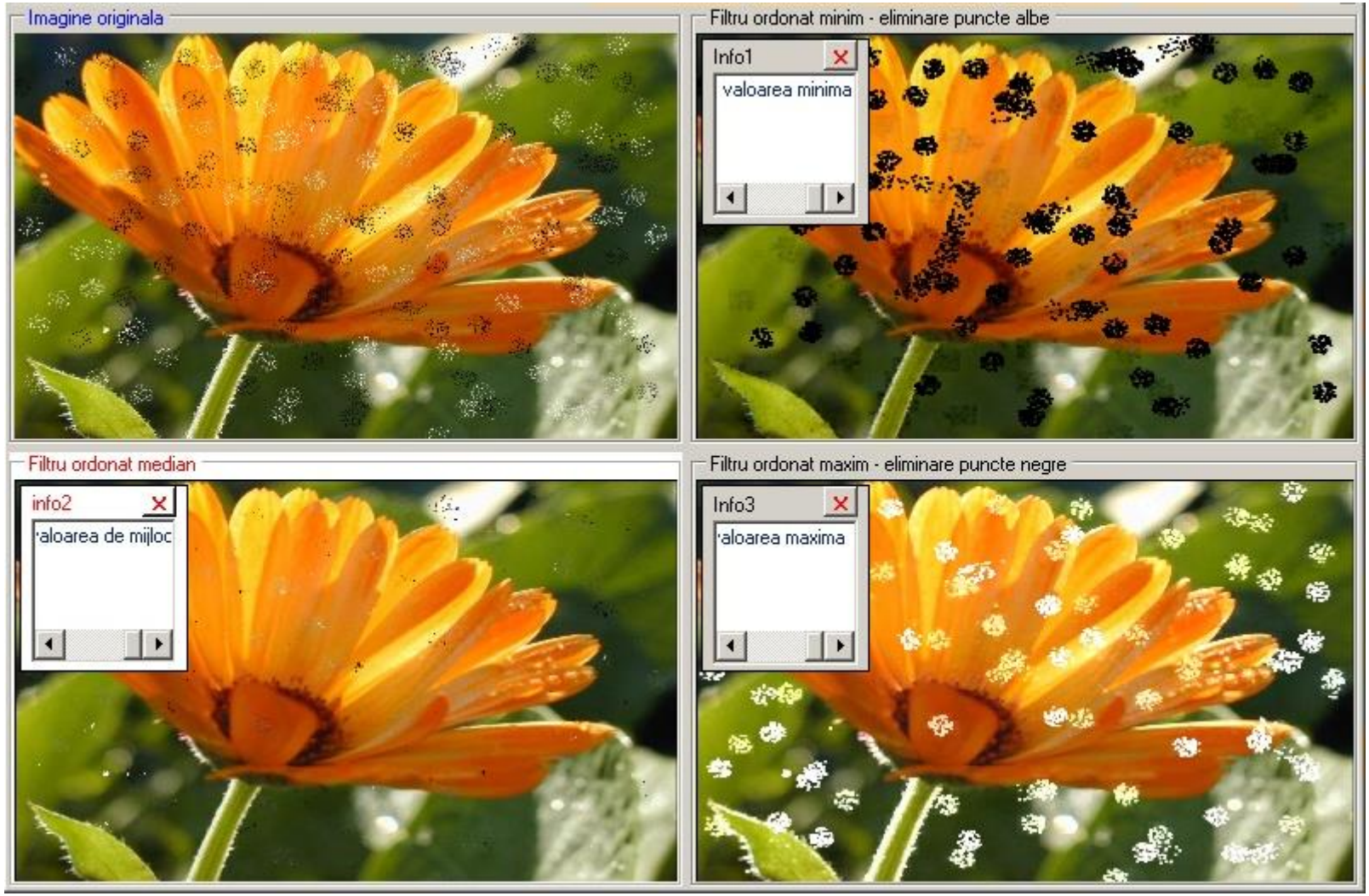
4) Netezeste oscilatiile de intensitate cu o perioada mai mica decat latimea ferestrei.

5) Modifica valoarea medie a intensitatilor din imagine daca distributia spatiala a zgomotului nu este simetrica in fereastra.

6) Conserva anumite forme de frontiere.

7) Nu genereaza noi nivele de gri.

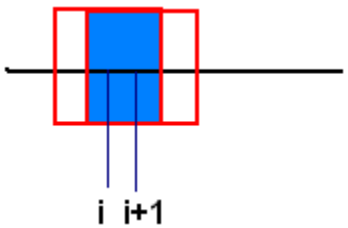
Filtre ordonate (4)



Implementarea filtrului median (1)

➢ Principalul efort de implementare: ordonarea vectorului de intensitati

1) Algoritmul Huang



- Fereastra de filtrare este deplasata pe randuri, de la stanga la drapta
- Fereastra are n linii si m coloane
- Ferestrele centrate in pixelii (i,j) si (i+1,j) au in comun (mxn -n) pixeli
- Calculul valorii mediane din fereastra curenta, centrata in (i+1,j), se bazeaza pe valoarea mediana a ferestrei centrate pe (i,j) si pe histograma imaginii din fereastra curenta.

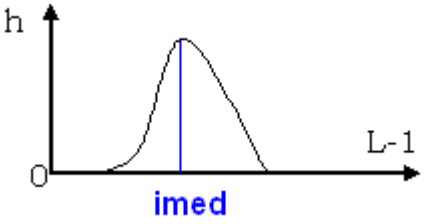
Fie imed valoarea mediana si N = nxm, numarul de pixeli din fereastra curenta

- N/2 pixeli ai imaginii din fereastră au intensitati mai mici decat imed.
- Numarul de pixeli din fereastră cu intensitatea mai mica decat imed este

$$\sum_{j=0}^{imed-1} h(j)$$

h este vectorul histograma (h(k)=numarul de pixeli cu intensitatea k)

- imed se obtine din conditia: $\sum_{j=0}^{imed-1} h(j) = N/2$



Implementarea filtrului median (2)

Algoritmul Huang

Pentru fiecare rand al imaginii

Pentru prima fereastra de pe rand

- calculeaza histograma in *vectorul h*
- construieste si sorteaza vectorul intensitatilor pixelilor din fereastra
- determina valoarea mediana, *imed*

Pentru celelalte ferestre de pe rand

- actualizeaza histograma (*vectorul h*):
  - scazand contributiile pixelilor de pe coloana din stanga ferestrei
  - adaugand contributiile pixelilor de pe ultima (noua) coloana a ferestrei
- actualizeaza *imed* prin incrementare/decrementare astfel incat:

$$\sum_{j=0}^{imed-1} h(j) = N/2$$

Observatie:

Filtrul nu se aplica pe marginea de latime n/2, respectiv m/2 a imaginii. Pentru filtrarea acestor margini se foloseste o fereastra trunchiata.

Implementarea filtrului median (3)

2) Calculul iterativ al valorii mediane, bit cu bit

Fie { 3,9,4,7,5} intensitatile pixelilor din fereastra.

3	9	4	7	5

1. Se determina bitul valoare medie de pe pozitia cea mai semnificativa a intensitatilor din fereastra.

   Acesta va fi bitul cel mai semnificativ al valorii mediane.

b3	b2	b1	b0
0	0	1	1
1	0	0	1
0	1	0	0
0	1	1	1
0	1	0	1

med = 0

2. Se continua numai cu valorile care au b3 = 0.

   Se determina bitul valoare medie din pozitia b2

b3	b2	b1	b0
0	0	1	1
1	0	0	1
0	1	0	0
0	1	1	1
0	1	0	1

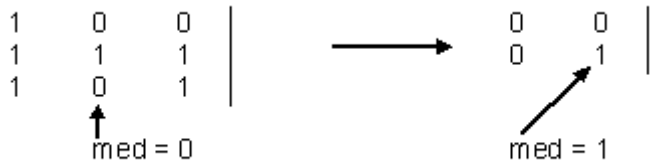
med = 0

0	0	1	1
0	1	0	0
0	1	1	1
0	1	0	1

med = 1

Implementarea filtrului median (4)

3. Se procedeaza la fel pentru a obtine bitii b1 si b0 ai valorii mediane



- Valoarea mediana ar putea fi 4 (alegand b0 = 0) sau 5 (b0 =1).

  Vectorul intensitatilor: { 3,9,4,7,5}

- N= 5, N/2 = 2 → Valoarea mediana trebuie sa fie mai mare (mica) decat doua elemente ale vectorului, deci se alege b0 =1.

- Atunci cand pe o pozitie numarul de biti = 0 este egal cu numarul de biti = 1, sunt necesare teste suplimentare pentru alegerea bitului valorii mediane. Testul foloseste conditia ca N/2 elemente ale vectorului sa fie mai mici ca valoarea mediana.

Filtrul de netezire conservativ (1)

- Filtru neliniar
- Netezeşte zgomotul conservând frontierele / detaliile reprezentate prin frecvențe mari în imagini.
- Nu introduce niveluri noi de intensitate.
- Proiectat pentru a elimina zgomotul de tip “sare şi piper”: pixelii izolați cu intensitate foarte mare sau foarte mică în comparație cu cea a pixelilor din vecinătatea lor.
- Mai puțin eficient pentru eliminarea zgomotului aditiv de tip Gaussian sau uniform.

Aplicarea filtrului

Fie: f imaginea de intrare şi g imaginea de ieşire;

Imin/Imax, cea mai mica/mare valoare de intensitate din vecinatatea pixelului (x,y)

daca Imin <= f(x,y) <= Imax

atunci g(x,y) = f(x,y)

altfel

daca f(x,y) > Imax atunci g(x,y) = Imax

daca f(x,y) < Imin atunci g(x,y) = Imin

Filtrul de netezire conservativ (2)

Exemplu:

Pixelii din fereastra de filtrare

120	125	130
126	**160**	125
126	128	125

Imax = 130, Imin = 120, f(x,y) =160
g(x,y) ← 130

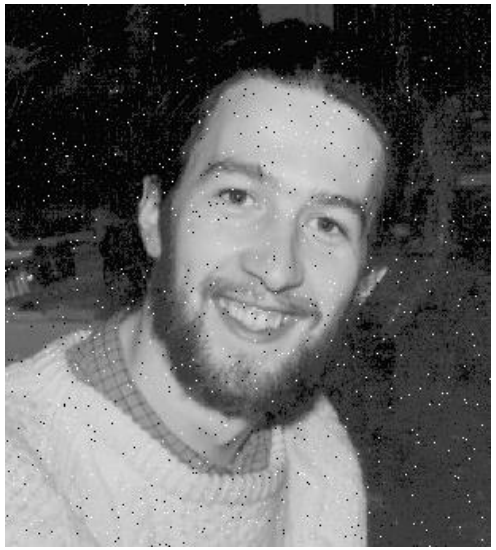
- Filtrul da rezultate bune pentru imagini cu nivel redus de zgomot “sare si piper”.
- Atunci cand mai multi pixeli din fereastra de filtrare sunt afectati de zgomot, filtrul este mai putin eficient.

➢ Filtrele liniare nu pot elimina total zgomotul impuls (sare si piper): un pixel de zgomot poate contribui semnificativ la suma ponderata a filtrului

➢ Filtrele neliniare (ex. filtrul median si filtrul conservativ) pot elimina in totalitate zgomotele impuls

➢ Implementarea filtrului conservativ: efort de calcul mai mic decat pt filtrul median!

Filtrul de netezire conservativ (3)



Dupa aplicarea filtrului medie

Dupa aplicarea filtrului median

Dupa aplicarea filtrului conservativ

- Filtrul conservativ este singurul care a pastrat reflexia luminii din ochii subiectului

- Conserva detaliile mai bine decat filtrul median

Restaurarea si Imbunatatirea imaginilor in domeniul spatial -2

Prof. univ. dr. ing. Florica Moldoveanu

Curs *Sisteme de Prelucrare Grafică* – UPB, Automatică şi Calculatoare
2021-2022

Corectia distorsiunilor geometrice(1)

❖ **Operație de restaurare a imaginilor digitale**

- **Surse ale distorsiunilor geometrice**:

  - Senzorul optic al sistemului de achizitie: distorsiuni optice, aspect ratio (scalare inegala pe verticala/orizontala), s.a (sunt multe tipuri de senzori).

  - Geometria vizualizarii : efectul panoramic (distorsiune la scara)

  - Mișcarea platformei (aeronavă / satelit) sau a țintei.

- Distorsiunea geometrică a unei imagini se manifestă printr-o deplasare a pozițiilor pixelilor din imagine față de pozițiile lor ideale, produse de un sensor ideal în condiții de achiziție ideale.

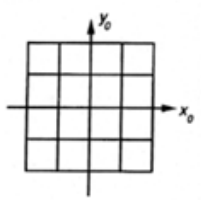
- **Corectia** se bazeaza pe cunoasterea modelului matematic al distorsiunilor.

Corectia distorsiunilor geometrice()

- Distorsiunea geometrica este definita printr-un set de ecuatii de transformare a pozitiilor pixelilor, din pozitiile reale (x,y) in pozitiile distorsionate (x',y'):

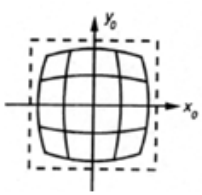
$$x' = h_x (x,y)$$
$$y' = h_y (x,y)$$

h_x si h_y sunt de obicei functii liniare pentru distorsiunile perspectiva si functii de grad 2 pentru distorsiunile produse de lentilele camerelor foto.



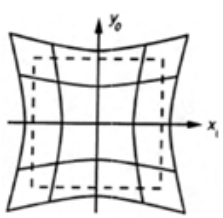
- fara distorsiune

Exemple de distorsiuni produse de lentilele camerelor



- distorsiune Barrel (butoi)

- imaginea este comprimata la colturi



- distorsiune Pincushion (pernita de ace)

- imaginea este extinsa la colturi

Corectia distorsiunilor geometrice(3)

f(x,y) imaginea de intrare, distorsionata

g(x,y) imaginea corectata.

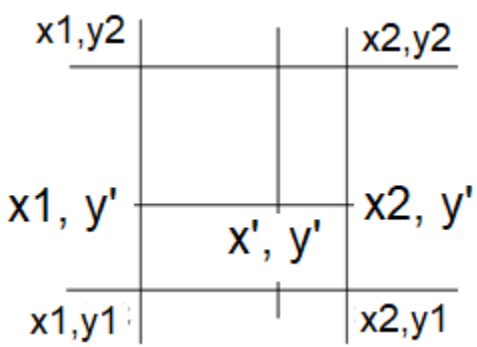
Corectia:

pentru fiecare pixel (x, y) al imaginii f(x,y)

Calculeaza pozitiile distorsionate: $x' = h_x(x,y)$ si $y' = h_y(x,y)$

g(x,y) = f (x', y') - memoreaza in pozitia corecta intensitatea din pozitia distorsionata

(x', y'): numere reale → g(x,y) se calculeaza prin interpolare biliniara intre valorile pixelilor din f



x1 < x' < x2 , y1 < y' < y2 , x2=x1+1, y2=y1+1

g(x,y) = f(x',y') se va aproxima astfel:

a = (y2 - y') * f(x1,y1) + (y'-y1) * f(x1,y2)

b = (y2 - y') * f(x2,y1) + (y'-y1) * f(x2,y2)

g(x,y) = (x2-x')* a + (x'-x1) * b

Estimarea zgomotului dintr-o imagine

- **Zgomotul (“noise”)** : informatie falsă într-o imagine discretă, introdusa de dispozitivele de achizitie a imaginii – pixeli a caror intensitate nu corespunde informatiei din imaginea reala

- **Eliminarea zgomotului:**
  - **operație de restaurare a imaginii**
  - **se realizeaza prin filtrarea imaginii în domeniul spatial sau al frecventei**

Estimarea zgomotului

Fie n imagini ale aceleiasi scene, obtinute cu acelasi sistem de achizitie si in aceleasi conditii de achizitie: E_0, ... E_{n-1} de N*N pixeli.

Pentru fiecare pixel (*i*, *j*) se calculeaza media intensitatilor sale din cele n imagini si deviatia standard:

$$\overline{E}(i,j) = \frac{1}{n}\sum_{k=0}^{n-1} E_k(i,j) \qquad \sigma(i,j) = \left(\frac{1}{n-1}\sum_{k=0}^{n-1}\left(\overline{E}(i,j) - E(i,j)\right)^2\right)^{\frac{1}{2}}$$

Media zgomotului pentru intreaga imagine este: $\dfrac{\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\sigma(i,j)}{N \times N}$

Zgomotul maxim este: $\max\limits_{\substack{0 \le i \le N-1 \\ 0 \le j \le N-1}}\left(\sigma(i,j)\right)$

Modelarea zgomotelor(1)

- Eliminarea zgomotului din imagini se bazeaza pe cunoasterea modelului matematic al diferitelor tipuri de zgomot.

Exista doua tipuri de zgomote:

- Independente de conţinutul imaginii
- Dependente de conţinutul imaginii

O imagine cu zgomot independent de conţinutul imaginii poate fi modelata prin:

$g(x,y) = f(x,y) + n(x,y)$ unde:

$f(x,y)$ este imaginea fara zgomot

$n(x,y)$ reprezinta zgomotul independent de continutul imaginii, numit si ***zgomot aditiv***

Zgomotul dependent de continutul imaginii (de exemplu, datorita radiatiilor monocromatice produse de o suprafata, care produc interferenţe de unde) poate fi reprezentat printr-un model ne-liniar: model mai complicat →se considera (daca se poate) zgomotul independent de continutul imaginii.

Modelarea zgomotelor(2)

❖ Cunoasterea modelelor diferitelor tipuri de zgomote permite eliminarea lor din imagini prin operatii de filtrare.

- Tipurile uzuale de zgomot:
  - zgomotul **uniform**
  - zgomotul **gaussian,**
  - zgomotul **sare si piper (salt-and peper)** numit si **impuls**

- Zgomotul modifica nivelul de intensitate (gri) al pixelilor in mod aleator.
  - ➢Se modeleaza ca o variabila aleatoare discretă cu o functie de probabilitate care depinde de tipul zgomotului.
  - ➢Pentru modelarea zgomotelor se folosesc functii de densitate a probabilitatii (FDP) (Probability Density Function: PDF).

Modelarea zgomotelor(3)

O variabila **x** este o **variabila aleatoare discreta** daca poate lua orice valoare dintr-un set de valori: X: [x1, x2, x3,…., xn].

❖**FDP(x): X →R functia de densitate a probabilitatii variabilei x**

FDP(xi), xi din X – probabilitatea ca variabila x sa aiba valoarea xi

❖**Valoarea medie a variabilei aleatoare discrete**:

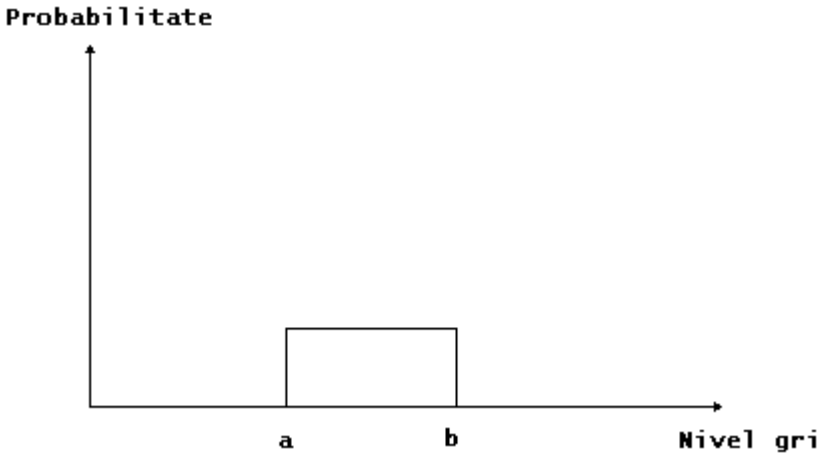
$\mu = \sum(x * FDP(x))$ cu x din X, FDP(x)>0

❖**Varianța variabilei aleatoare discrete** (Imprastierea valorilor – distanta lor fata de medie)

$\sigma^2 = \sum(x - \mu)^2 FDP(x)$ cu x din X, FDP(x)>0

Zgomotul uniform

- Zgomot cauzat de operatia de cuantizare cu un pas constant (reprezentarea valorilor esantioanelor pe un numar limitat de niveluri discrete): valorile zgomotului sunt egal distribuite în intervalul de intensitati [a,b] rezultate prin cuantizare, de ex. 0-255.



Functia de densitate a probabilitatii este:

FDP(g) = 1/(b-a), a<=g<=b

FDP(g) = 0, altfel

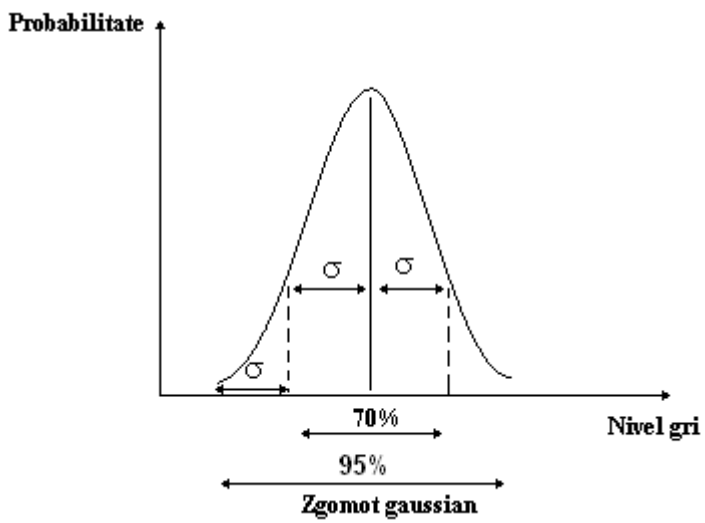
Ex: pt 0 <= g <= 255
FDP (g) = 1.0/255

Intensitatea unui pixel dintr-o imagine cu zgomot uniform:

g(x,y) = f(x,y) + FDP(f(x,y))

FDP(f(x,y)) are aceeasi valoare pt orice f(x,y).

Zgomotul Gaussian (1)



Zgomotul Gaussian este modelat prin functia Gaussiana de densitate a probabilitatii:

$$FDP(g) = (1/(2\pi\sigma)^{1/2}) \exp(-(g-m)^2/2\sigma^2)$$

unde: g este un nivel de gri;
m = media zgomotului;
σ = deviatia standard a zgomotului;

FDP este nula la aprox. 3σ de valoarea medie.

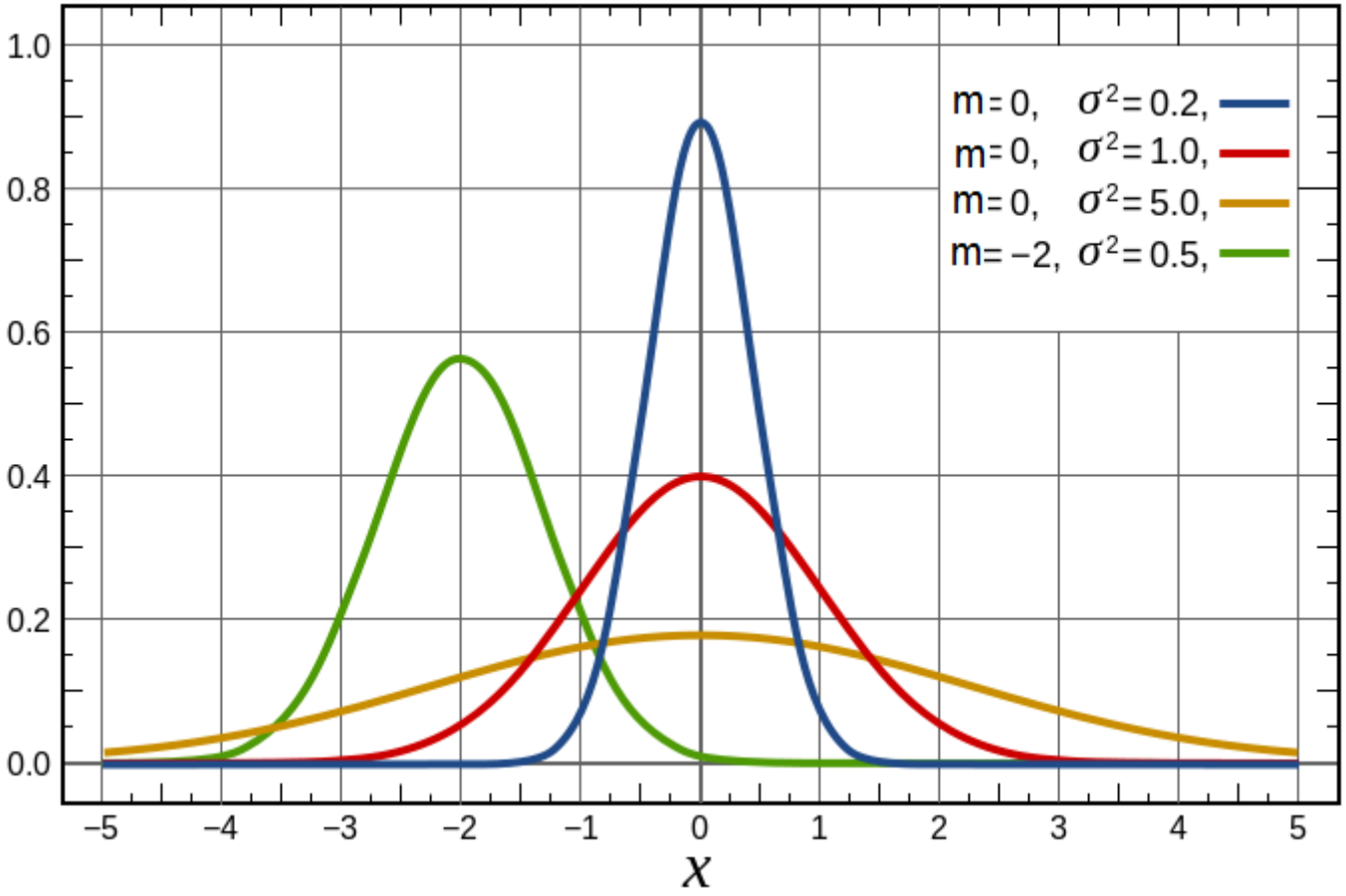
Fiecare pixel din imaginea cu zgomot Gaussian are o valoare rezultata din insumarea valorii reale a pixelului cu o valoare aleatoare cu distributie Gaussiana.

$g(x,y) = f(x,y) + FDP(f(x,y))$

- *Modelul gaussian* este cel mai des folosit pentru modelarea proceselor cu zgomot natural, cum este zgomotul produs de sistemul de achizitie al imaginilor: zgomotul senzorului, produs de proasta iluminare sau temperatura ridicata, zgomotul introdus de circuitele electronice.

Zgomotul Gaussian (2)

Functia Gaussiana de densitate a probabilitatii



m: media distributiei

σ^2 :variatia distributiei

Zgomotul Gaussian (3)

Imaginea originala: f(x,y)

f(x,y) + FDP1(f(x,y))

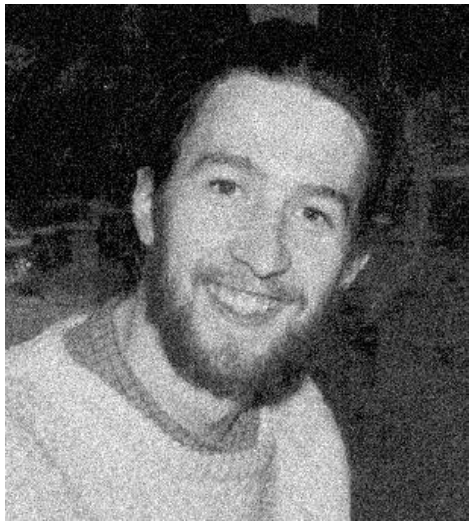
f(x,y) + FDP2(f(x,y))

Imagini alterate cu zgomot Gaussian:

FPD1: $\sigma = 8$

FPD2: $\sigma = 13$

FPD3: $\sigma = 20$

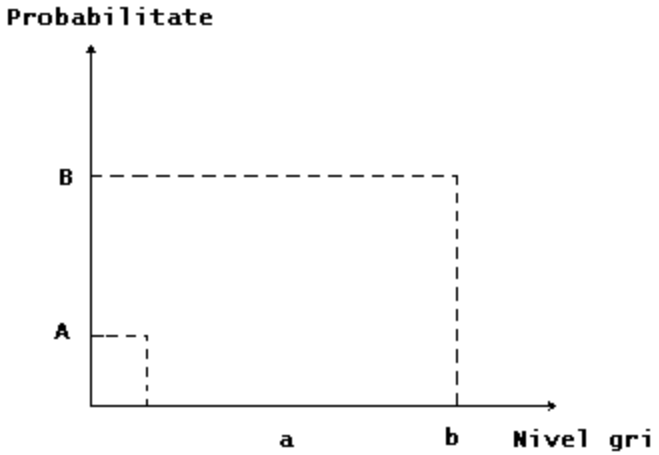


f(x,y) + FDP3(f(x,y))

https://homepages.inf.ed.ac.uk/rbf/HIPR2/noise.htm

Zgomotul sare şi piper (salt-and-peper)

- Numit si **zgomot impuls** – produs de variatii bruste ale semnalului electric, cauzate de functionarea proasta a celulelor din senzorii camerelor, greseli ale locatiilor de memorie, erori de sincronizare in procesul de digitizare sau erori (pierderi de biti) pe canalul de comunicatie in cazul transmisiei imaginilor.
- Se manifesta prin puncte albe si negre cu aparitie aleatoare.



Functia de distributie a probabilitatii este:

FDP(g) =A, pentru g=a (pepper)
FDP(g) =B, pentru g=b (salt)
FDP(g) =0, altfel

g(x,y) = f(x,y) + FDP(f(x,y))

Pentru o imagine cu 256 nivele de intensitate: a este aproape de 0, b este aproape de 255.

Zgomot “piper”: puncte negre in zone de intensitate mare

Zgomot “sare”: puncte albe in zone de intensitate mica

Alte tipuri de zgomote

Alte tipuri de zgomote, denumite dupa tipul FDP:

- zgomot Rayleigh
- zgomot Gamma
- zgomot Exponential negativ

https://arxiv.org/ftp/arxiv/papers/1505/1505.03489.pdf

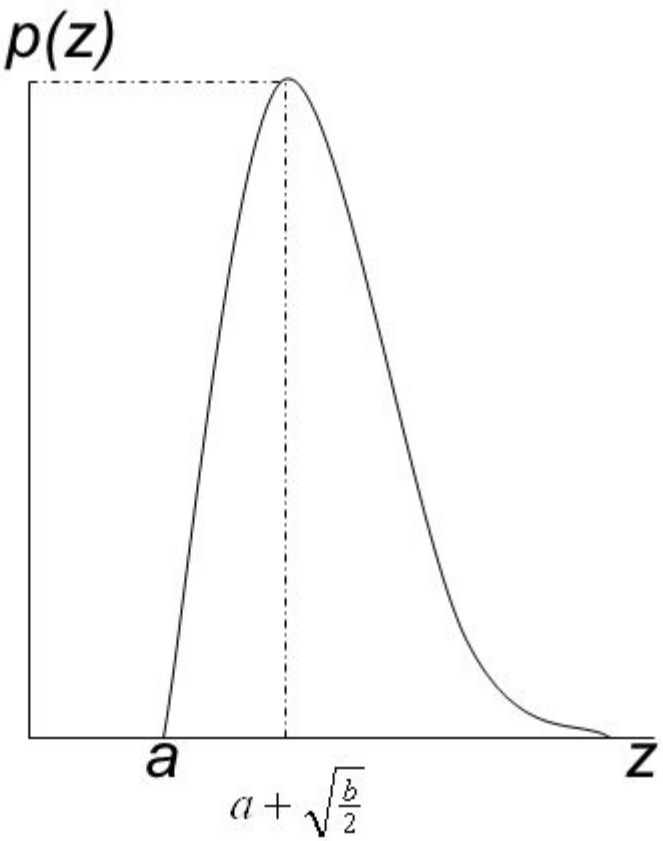
Rayleigh noise

- The PDF of a Rayleigh noise is given by

$$p(z) = \begin{cases} \frac{2}{b}(z-a)e^{-(z-a)^2/b} & for\ z \geq a \\ 0 & for\ z < a \end{cases}$$

The mean and variance are given

$$\mu = a + \sqrt{\pi b / 4} \quad \text{and} \quad \sigma^2 = \frac{b(4-\pi)}{4}$$

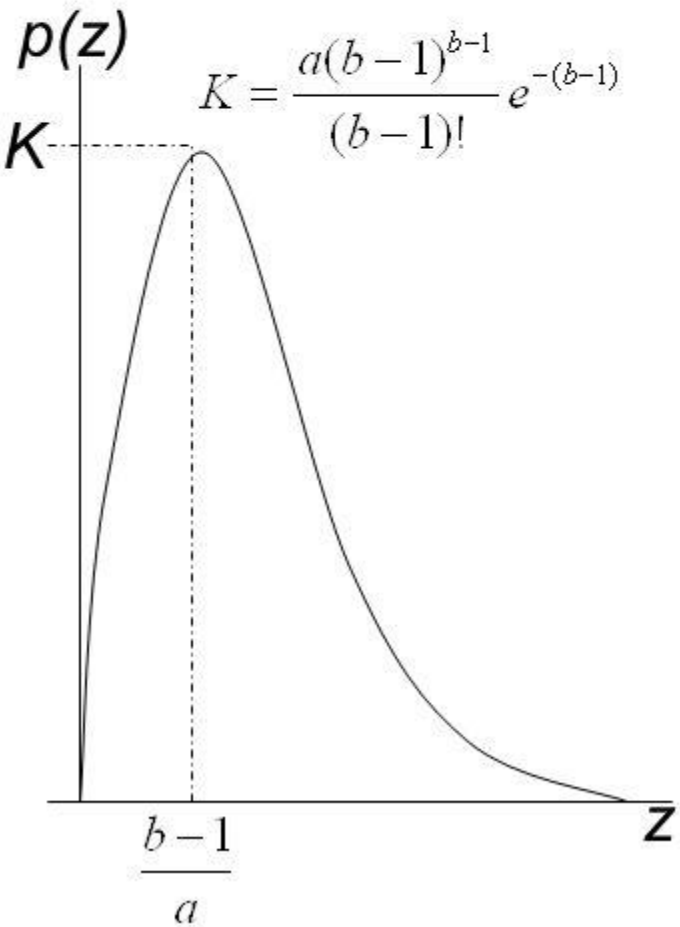


Apare în: imagini MRI, imagini subacvatice

Gamma noise

- The PDF of a Gamma noise is given by

$$p(z) = \begin{cases} \dfrac{a^b z^{b-1}}{(b-1)!} e^{-az} & for\ z \geq 0 \\ 0 & for\ z < 0 \end{cases}$$



The mean and variance are given

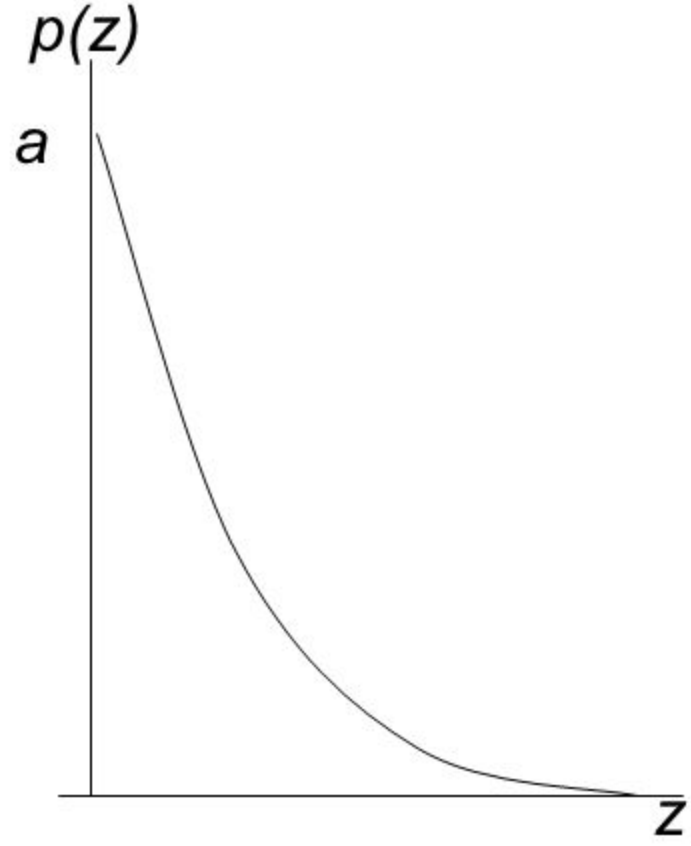
$$\mu = \frac{b}{a} \quad \text{and} \quad \sigma^2 = \frac{b}{a^2}$$

Apare în imaginile obtinute prin raze laser

Exponential noise

- The PDF of a Exponential noise is given by

$$p(z) = \begin{cases} ae^{-az} & for\ z \geq 0 \\ 0 & for\ z < 0 \end{cases}$$



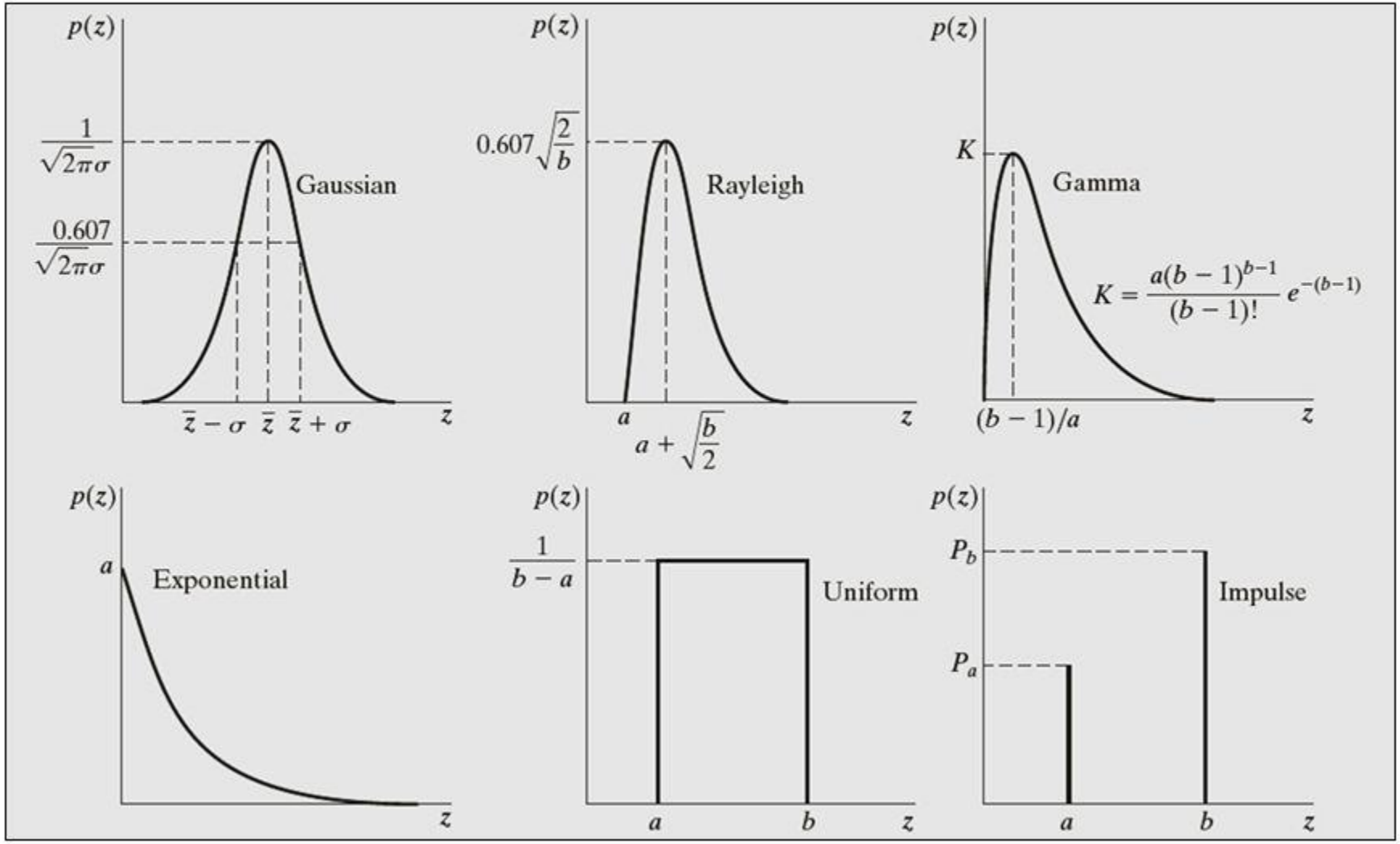
The mean and variance are given

$$\mu = \frac{1}{a} \quad \text{and} \quad \sigma^2 = \frac{1}{a^2}$$

Este un caz particular al functiei Gamma PDF,

in care b=1

Summary of different noise models



https://slideplayer.com/slide/8223642/

Detectia pixelilor de frontiera din imagini (Edge detection)

Prof. univ. dr. ing. Florica Moldoveanu

Detectia frontierelor din imagini (Edge detection)

Frontierele:

- Alcatuite din pixeli in care intensitatea variaza brusc.
- Pot reprezenta obiecte subtiri (linii) sau pixelii aflati la adiacenta a doua zone din imagine cu intensitati diferite.
- Un pixel de frontiera este numit pe scurt ***pixel de front*** (margine) sau ***front.***

Majoritatea metodelor de extragere a frontierelor presupun efectuarea urmatoarelor etape:

1. Detectia pixelilor de front.

Un **pixel de front** este un pixel in care intensitatea se schimba brusc: poate fi un punct de frontiera sau un punct de zgomot.

→ este utila eliminarea/reducerea zgomotelor din imagine, inainte de detectia pixelilor de front

→ filtrele de netezire diminueaza nu numai intensitatea punctelor de zgomot ci si pe aceea a pixelilor de frontiera!

2. Eliminarea pixelilor de front falsi.

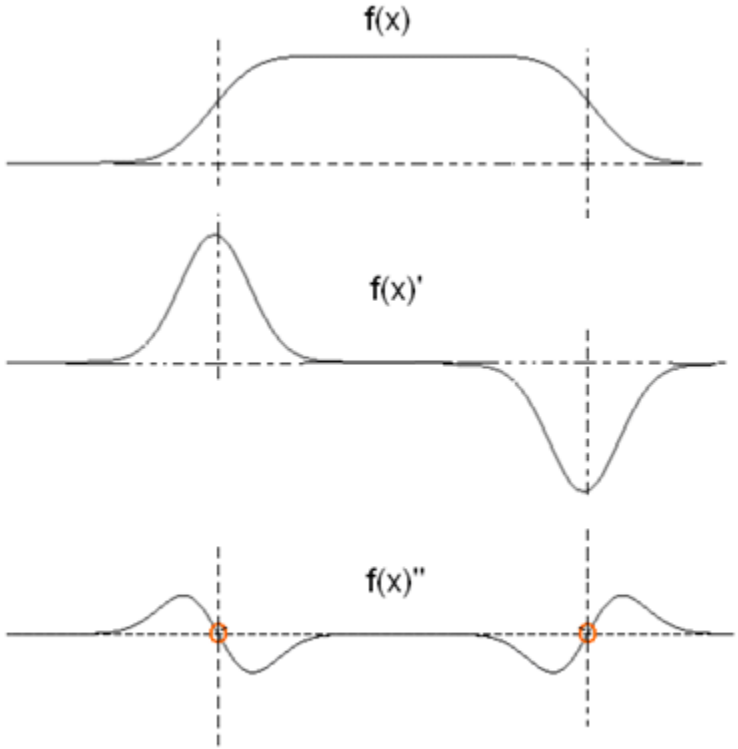
3. Conectarea pixelilor de front pentru a forma frontiere.

Detectia pixelilor de front(1)

Semnal uni-dimensional

Derivata de ordinul 1

Derivata de ordinul 2



Profilul 1D al unui front

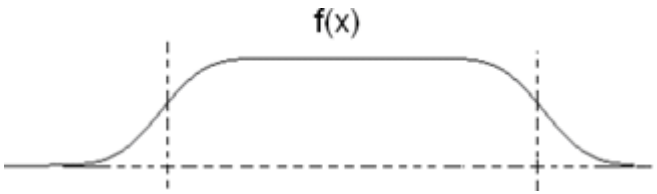
Pentru detectia pixelilor de front:

- Se utilizeaza operatori (masti de convolutie) care aproximeaza in planul discret Gradientul (derivata de ordinul 1) sau Laplacianul (derivata de ordinul 2 a) functiei imagine.
- Mastile sunt de dimensiune mica (3x3) → operatorii sunt sensibili la zgomot (pot produce puncte de front false).

Detectia pixelilor de front(2)

Bazata pe Gradientul imaginii:

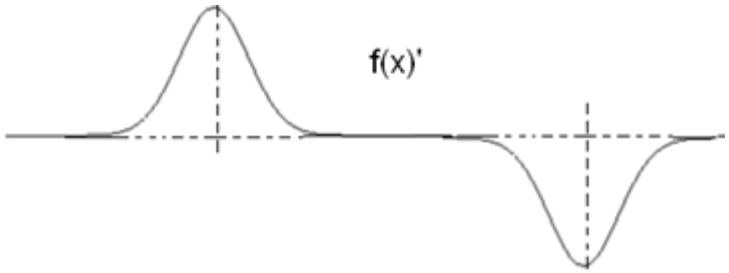
Amplitudinea gradientului



- Este zero in zonele de intensitate constanta
- **Este mare** in pixelii in care intensitatea se schimba brusc (**pixeli de front**)

Directia gradientului = directia frontului:

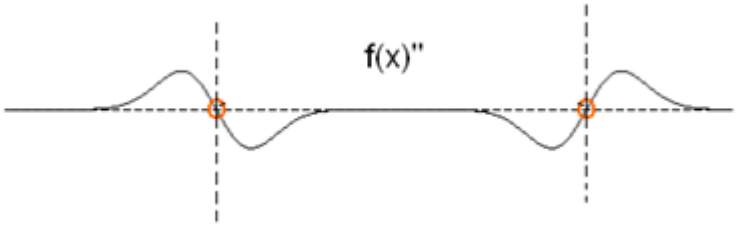
directia schimbarii maxime de intensitate



Bazata pe Laplacianul imaginii:

Laplacianul unei imagini contine:

- **Valori mari, pozitive si negative** in vecinatatea unui punct in care intensitatea se schimba brusc (**punct de front**)
- Zero in zone de intensitate constanta



Detectia pixelilor de front bazata pe Gradient-ul imaginii(1)

Vectorul gradient calculat intr-un pixel al unei imagini reprezinta directia si marimea variatiei maxime de intensitate in acel pixel.

Este definit în funcţie de derivatele parţiale ale funcţiei imagine, f(x,y):

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} \qquad D_x = \frac{\partial f(x,y)}{\partial x} \text{ şi } D_y = \frac{\partial f(x,y)}{\partial y}, \text{ atunci } \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} D_x \\ D_y \end{bmatrix}$$

***Amplitudinea* gradientului în pixelul (x,y) este**:

$$D(x, y) = (Dx^2 + Dy^2)^{0.5}$$

Ea reprezinta **puterea frontului** (marimea variatiei de intensitate) în pixelul respectiv

*Este **aproximata prin**:*

$$D(x, y) = |Dx| + |Dy| \qquad \text{sau} \qquad D(x, y) = \max(|Dx|, |Dy|)$$

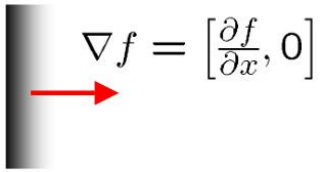
Detectia pixelilor de front bazata pe Gradientul imaginii(2)

***Direcţia* gradientului** (frontului) este:

θ = arctg(Dy/Dx) pentru Dx # 0

+/- 90, pentru Dx =0, Dy#0

0, pentru Dy=0, Dx#0



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$

$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

- **Directia gradientului este directia creşterii maxime de intensitate, intr-un pixel**
- Vectorul gradient este perpendicular pe directia frontierei.

Punctele de front dintr-o imagine sunt detectate calculând gradientul in fiecare pixel şi identificând acei pixeli pentru care amplitudinea gradientului este mai mare decât un *prag* dat.

Aproximarea discreta a gradientului

Aproximarea prin diferente finite

$$Dx = \frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+\Delta x, y) - f(x,y)}{\Delta x} \qquad Dy = \frac{\partial f(x,y)}{\partial y} \approx \frac{f(x, y+\Delta y) - f(x,y)}{\Delta y}$$

Măşti de convolutie

$\Delta x = \Delta y = 1$ ⟹ $Dx = f(x+1,y) - f(x,y)$

$Dy = f(x,y+1) - f(x,y)$

1	
-1	1

(x,y)

$[-1 \quad 1]$ $\quad \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

$$Dx \approx \frac{f(x+\Delta x, y) - f(x-\Delta x, y)}{2(\Delta x)} \qquad Dy \approx \frac{f(x, y+\Delta y) - f(x, y-\Delta y)}{2(\Delta y)}$$

	1	
-1	0	1
	-1	

½[-1 0 1] $\quad 1/2 \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$

Mastile detectorului bazat pe diferente finite: 2 pixeli → detector f. sensibil la zgomot

Cei mai utilizati detectori de fronturi bazati pe gradient sunt:

Roberts (numit si operatorul *cruce*), Sobel si Prewitt.

Detectorul Roberts(1)

Fie P1=f(x,y+1), P2=f(x+1,y+1), P3=f(x,y), P4=f(x+1, y)

P1	P2
P3	P4

(x,y)

Detectorul se aplica in pixelul (x,y):

D+ (x,y) = f(x + 1, y + 1) – f(x,y) = P2 - P3

D- (x,y) = f(x , y + 1) – f(x + 1, y) = P1 - P4

D+ si D- se calculeaza cu urmatoarele 2 masti de convolutie:

0	+1
-1	0

D +

+ 1	0
0	-1

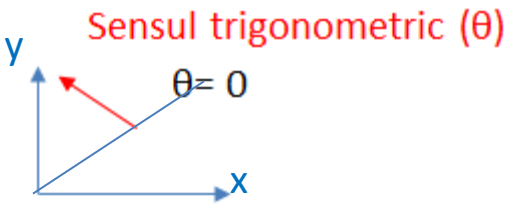
D-

Cele 2 masti sunt proiectate pentru detectia maxima a frontierelor cu directii de 45 (D+ =0) si 135 (D-=0) grade (directii perpendiculare, de unde si numele de “operatorul cruce”).

Amplitudinea gradientului in (x,y) este aproximata prin: D(x,y) = |D+(x,y)| + |D-(x,y)|

Orientarea frontului (directia gradientului) intr-un pixel (x,y) se poate calcula in functie de D+ si D-, tinand cont de orientarea grilei de pixeli.

Detectorul Roberts(2)



u = arctg(D+/D-) pentru D- # 0

+/- 90, pentru D- =0, D+ # 0

0, pentru D+=0, D- # 0

Unghiul 0 corespunde la D+=0 → θ = 45 grade

Directia frontului: (arctg(D+/D-) + 45) grade

Daca se calculeaza numai matricea amplitudinilor se foloseste masca:

+ 1	+1
-1	-1

P1	P2
P3	P4

Aproximarea amplitudinii:

D(x,y) = | P1 – P4| + |P2 – P3|

Principalul avantaj al utilizarii acestui detector este simplitatea calculelor: operatii de adunare si scadere cu valorile a numai 4 pixeli.

Dezavantajul: deoarece utilizeaza un nucleu foarte mic,

- este foarte sensibil la zgomot,
- raspunsul său la frontiere reale este slab daca acestea nu sunt foarte “abrupte” (o tranzitie de intensitate foarte mare).

Detectorul Sobel(1)

Gradientul este aproximat prin convolutia imaginii cu urmatoarele masti:

Dx:

-1	0	+1
-2	0	+2
-1	0	+1

Dy:

+1	+2	+1
0	0	0
-1	-2	-1

Pixelul (x,y) in care se calculeaza gradientul este cel pe care este centrata masca.

Daca I este imaginea de intrare:

Dx (x, y) = I(x+1, y+1) + 2* I(x+1, y) + I(x+1, y-1) – I(x-1, y+1) – 2* I(x-1, y) – I(x-1, y-1)

Dy (x, y) = I(x+1, y+1) + 2* I(x, y+1) + I(x-1, y+1) – I(x+1, y-1) – 2* I(x, y-1) – I(x-1, y-1)

Mastile sunt proiectate pentru raspuns maxim la frontiere verticale si orizontale.

Masca Dx evidentiaza frontierele orizontale: (Dx=0) iar Dy frontierele verticale (Dy=0)

Amplitudinea gradientului este aproximata prin: D(x,y) = |Dx| + |Dy|

Detectorul Sobel(2)

Directia gradientului:

θ= arctg (Dy / Dx), pentru Dx#0

+/- 90, pentru Dx =0, Dy#0 → frontiera orizontala, vectorul gradient vertical

0, pentru Dy=0, Dx#0 → frontiera verticala, vectorul gradient orizontal

θ =0 atunci cand directia contrastului maxim este de la stanga la dreapta (frontiera verticala)

Unghiurile sunt masurate in sens trigonometric fata de unghiul zero.

Daca se calculeaza numai matricea amplitudinilor se foloseste o masca de 3x3:

D(x,y) = | (P3 + 2xP6 + P9) – (P1 + 2x P4 + P7)| + | (P1 + 2xP2 + P3) – (P7 + 2x P8 + P9) |

P_1	P_2	P_3
P_4	P_5	P_6
P_7	P_8	P_9

Exemplu: aplicarea detectorului Sobel

Imaginea

0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0

Sobel

Dx

-1	0	1
-2	0	2
-1	0	1

Dy

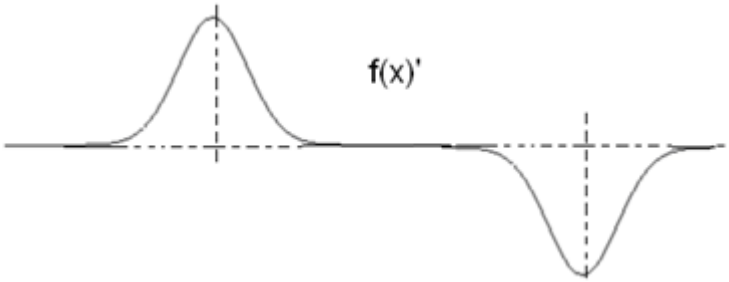
1	2	1
0	0	0
-1	-2	-1

Dx

0	1	2	3	4	5	6	7	8	9	10
	0	0	~~400~~	400	0	0	-400	-400	0	
	0	0	~~400~~	400	0	0	-400	-400	0	
	0	0	400	~~400~~	0	0	-400	-400	0	
	0	0	400	400	0	0	-400	-400	0	
	0	0	400	400	0	0	-400	-400	0	
	0	0	400	400	0	0	-400	-400	0	

Dy=0, Dx>0 → θ =0

Dy=0, Dx<0 → θ =180



- Un pixel (x,y) este "pixel de front" daca ampitudinea gradientului în acel pixel, D(x,y) (|Dx|+|Dy|), este peste o valoare de prag.

- Frontierele produse de operatorul Sobel sunt groase (mai mult de un pixel).

Detectorul Sobel(3)

Comparatie cu Roberts

- Necesita mai multe calcule
- Masca fiind mai mare, este mai putin sensibil la zgomot
- Produce valori de amplitudine mai mari decat cele produse de operatorul Roberts, pentru aceleasi frontiere.

Vizualizarea matricei amplitudinilor

- Matricea amplitudinilor, $D(x,y) = |Dx| + |Dy|$, poate fi vizualizata ca o imagine in nivele de gri.

  Daca $D(x,y) >$ nivelul de intensitate maxim

  atunci $D(x, y) \leftarrow$ nivelul de intensitate maxim.

- Frontierele apar adesea in aceasta imagine ca linii a caror latime este de mai multi pixeli.

Pentru subtierea frontierei este necesara o postprocesare, cum ar fi aceea de “histerezis”, din algoritmul Canny.

Detectorul Prewitt

Gradientul este aproximat prin convolutia imaginii cu urmatoarele masti:

Dx:

-1	0	+1
-1	0	+1
-1	0	+1

Dy:

+1	+1	+1
0	0	0
-1	-1	-1

Amplitudinea gradientului este aproximata prin: $D(x,y) = |Dx| + |Dy|$

Mastile sunt proiectate pentru raspuns maxim la frontiere verticale si orizontale.

Directia gradientului:

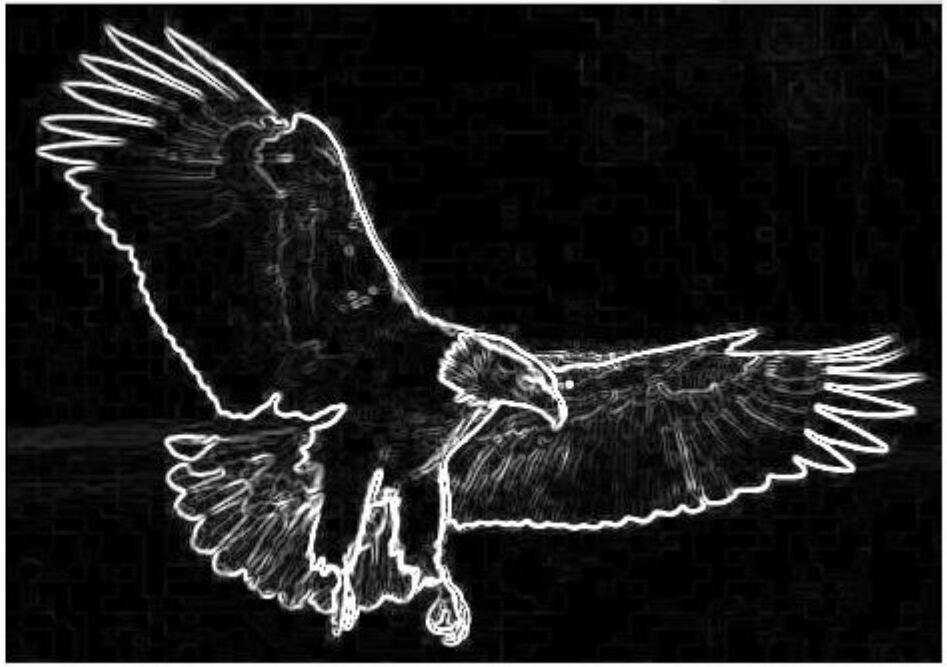
θ= arctg (Dy / Dx), pentru Dx#0

+/- 90, pentru Dx =0, Dy#0

0, pentru Dy=0, Dx#0

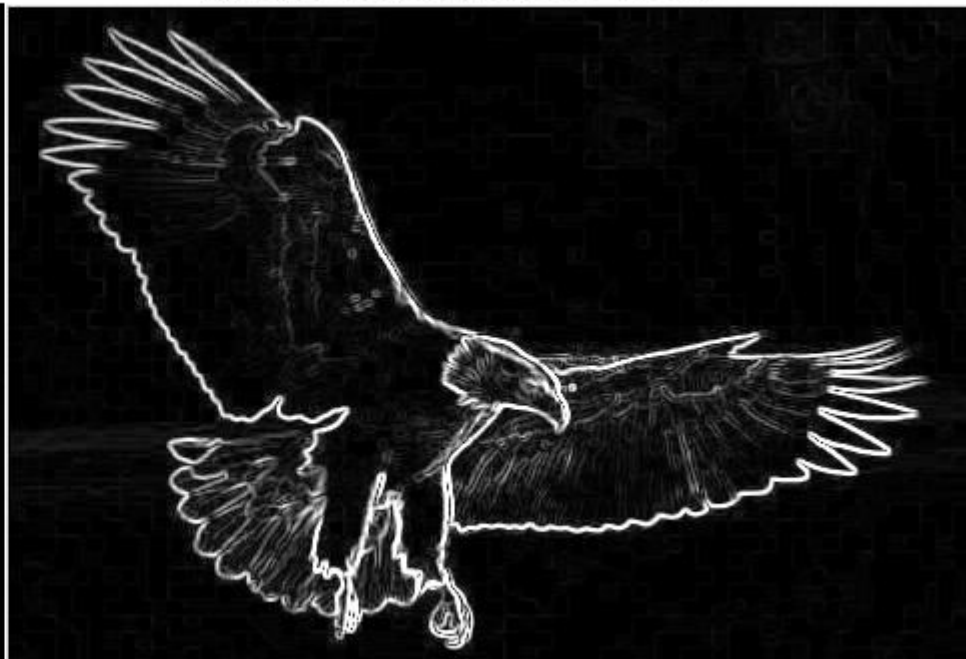
Imaginea originala

Amplitudinea Gradientului - Sobel



Amplitudinea Gradientului - Roberts

Amplitudinea Gradientului - Prewitt



Detectia pixelilor de front bazata pe Laplacianul imaginii(1)

Operatorul Laplacian este definit în funcţie de derivatele parţiale de ordinul 2 :

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2}(x,y) + \frac{\partial^2 f}{\partial y^2}(x,y)$$

Derivatele parţiale de ordinul 2, de-a lungul direcţiilor x si y, sunt aproximate prin diferenţe finite astfel:

$$\frac{\partial^2 f}{\partial x^2} = \frac{(f(x+\Delta x, y) - f(x,y)) - (f(x,y) - f(x-\Delta x, y))}{(\Delta x)^2} \qquad \frac{\partial^2 f}{\partial y^2} = \frac{(f(x, y+\Delta y) - f(x,y)) - (f(x,y) - f(x, y-\Delta y))}{(\Delta y)^2}$$

Pentru $\Delta x = \Delta y = 1$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) - 2f(x,y) + f(x-1, y).$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) - 2f(x,y) + f(x, y-1).$$

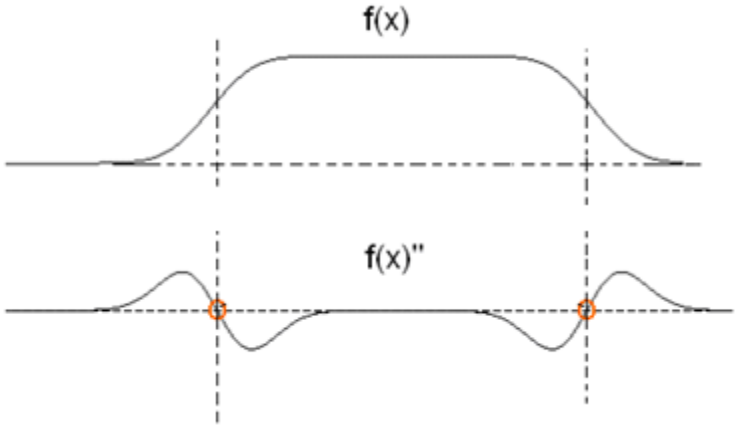
$$\nabla^2 f = f(x+1, y) - 4f(x,y) + f(x-1, y) + f(x, y+1) + f(x, y-1).$$

Detectia pixelilor de front bazata pe Laplacianul imaginii(2)

Rezulta ca Laplacianul poate fi implementat prin urmatoarea masca de convolutie:

0	1	0
1	-4	1
0	1	0

$$\nabla^2 f = f(x+1,y) - 4f(x,y) + f(x-1,y) + f(x,y+1) + f(x,y-1).$$



Fie f(x,y) imaginea de intrare, in mai multe nivele de gri;

g(x, y) – imaginea obtinuta prin aplicarea operatorului Laplacian, este tot o imagine cu mai multe nivele de gri:

g(x,y) = 0 in zonele de intensitate constanta

<0 sau > 0 in vecinatatea unei frontiere

(>0 de partea mai intunecata, <0 de partea mai luminoasa)

Pentru afisarea iesirii operatorului Laplacian sub forma unei imagini, valorile din |g(x,y)| sunt normalizate la domeniul [0-nivel maxim de intensitate].

Exemplu: aplicarea detectorului Laplacian

Imaginea

0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0

Laplacian

0	1	0
1	-4	1
0	1	0

Laplacianul imaginii

0	1	2	3	4	5	6	7	8	9	10
	0	0	100	-100	0	0	-100	100	0	
	0	0	100	-100	0	0	-100	100	0	
	0	0	100	-100	0	0	-100	100	0	
	0	0	100	-100	0	0	-100	100	0	
	0	0	100	-100	0	0	-100	100	0	
	0	0	100	-100	0	0	-100	100	0	

g(x,y) = 0 in zonele de intensitate constanta

<0 sau > 0 in vecinatatea unei frontiere

(>0 de partea mai intunecata, <0 de partea mai luminoasa)

Nota: frontierele din exemplu sunt ideale!

Imaginea originala

Laplacianul imaginii

Extragerea frontierelor bazata pe Laplacianul imaginii(3)

Pentru extragerea frontierelor din Laplacianul imaginii, se poate proceda in mai multe moduri:

1) Se aplica un prag peste |g(x,y)|, rezultand o imagine binara, B(x,y):

B(x,y) = 1 , daca |g(x,y)| > prag

0, altfel

Cat de aproape de frontiera reala sunt pixelii cu valoarea 1 in B(x,y) depinde de alegerea pragului.

2) Sunt considerate puncte de frontiera acele puncte in care g(x,y) # 0 si amplitudinea gradientului este peste un prag dat.

3) Daca in vecinatatea pixelului curent valorile Laplacianului sunt atat pozitive cat si negative, dar peste o valoare de prag ca marime, atunci pixelul este unul de frontiera (trecere prin zero a Laplacianului).

Pixelii de frontiera sunt setati la nivelul maxim de intensitate, restul la zero.

Aceasta metoda este redata in pseudocodul care urmeaza:

Detectia frontierelor folosind Laplacianul(4)

```
// in: matricea continand Laplacianul unei imagini de mxn pixeli
// 2*L+1: mărimea vecinatatii (L impar);
// out: matricea de iesire, in care pixelii de frontiera sunt setati la intensitatea maxima, restul la
// valoarea 0.
// prag: valoarea de prag; F este cea mai mare valoare reala.
for ( i=L; i<m-L; i++) // pentru fiecare element al matricei de intrare
    for( j=L; j< n-L; j++)
    {       min = F; max = - min;
            for( k=-L; k<= L; k++) // calculeaza val. min. si max. din vecinatatea elementului curent
                for(l=-L; l<=L; l++)
                {val = in[i+k][j+l];
                 if(val < min) min = val; if( val > max) max = val;
                }
            if(min < - prag && max > prag) out[i][j] = (unsigned char) 255;
            else out[i][j] =0; // sunt eliminate falsele puncte de frontiera (zgomote)
    }
```

Utilizarea Laplacianului ca filtru

- **Masca de convolutie Laplacian poate fi utilizata ca filtru de îmbunătățire a imaginilor prin evidentierea frontierelor** (image sharpening).
- Mastile de convolutie a, b, c, produc efecte asemanatoare cu masca Laplacian.

0	1	0
1	-4	1
0	1	0

Laplacian

a

0	–1	0
–1	4	–1
0	–1	0

b

–1	–1	–1
–1	8	–1
–1	–1	–1

c

1	1	1
1	-8	1
1	1	1

Obs: filtrele nu modifica valorile pixelilor din zonele de intensitate constanta

- O imagine cu frontierele evidentiate, g'(x,y), se poate obtine si astfel:

- Daca g(x,y) este iesirea operatorului Laplacian sau rezultatul aplicarii mastii (c), atunci:

  $g'(x,y) = f(x,y) - g(x, y)$; daca $g'(x,y)<0$ atunci $g'(x,y) =0$

- Daca g(x,y) este rezultatul aplicarii uneia dintre mastile (a) sau (b) atunci:

  $g'(x,y) = f(x,y) + g(x,y)$ daca $g'(x,y)<0$ atunci $g'(x,y) =0$

Evidentierea frontierelor

0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0
0	0	0	0	100	100	100	100	0	0	0

Imaginea de intrare: f(x,y)

Extragerea frontierelor

Laplacianul imaginii: g(x,y)

0	1	2	3	4	5	6	7	8	9	10
	0	0	100	-100	0	0	-100	100	0	
	0	0	100	-100	0	0	-100	100	0	
	0	0	100	-100	0	0	-100	100	0	
	0	0	100	-100	0	0	-100	100	0	
	0	0	100	-100	0	0	-100	100	0	
	0	0	100	-100	0	0	-100	100	0	

Convolutia imaginii cu masca (c): h(x,y)

2	3	4	5	6	7	8	9	10
0	300	-300	0	0	-300	300	0	0
0	300	-300	0	0	-300	300	0	0
0	300	-300	0	0	-300	300	0	0

Imbunatatirea imaginii prin evidentierea frontierelor

g'(x,y)= f(x,y) – g(x, y)

2	3	4	5	6	7	8	9	10
0	0	200	100	100	200	0	0	0
0	0	200	100	100	200	0	0	0
0	0	200	100	100	200	0	0	0

daca g'(x,y)<0 atunci g'(x,y) =0

g'(x,y)=f(x,y) – h(x, y)

2	3	4	5	6	7	8	9	10
0	0	400	100	100	400	0	0	0
0	0	400	100	100	400	0	0	0
0	0	400	100	100	400	0	0	0

Operatorul LoG - Laplacian of Gaussian (1)

Laplacianul produce variatii pozitiv/negativ si la schimbari mici de intensitate, care nu reprezinta frontiere, sau in pixeli de zgomot. De aceea, inainte de Laplacian se poate aplica imaginii un filtru.

LOG: Laplacian of Gaussian

Imbina urmatoarele doua operatii:

- Filtrarea, prin aplicarea unui filtru Gaussian
- Detectia frontierelor, ca puncte de trecere prin zero a Laplacianului

Filtrul Gaussian: $g(x,y) = (1/(2\pi\sigma^2)) \exp(-(x^2 + y^2)/(2\sigma^2))$

$$g(x,y) \bullet f(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} w(i,j) * f(x+i, y+j)$$

Convolutia discreta a functiei imagine cu filtrul Gaussian

w este masca de convolutie care aproximeaza functia g(x,y) in planul discret

Laplacianul convolutiei celor doua functii este: $\nabla^2(g(x,y) \bullet f(x,y)) = \nabla^2(g(x,y)) \bullet f(x,y)$

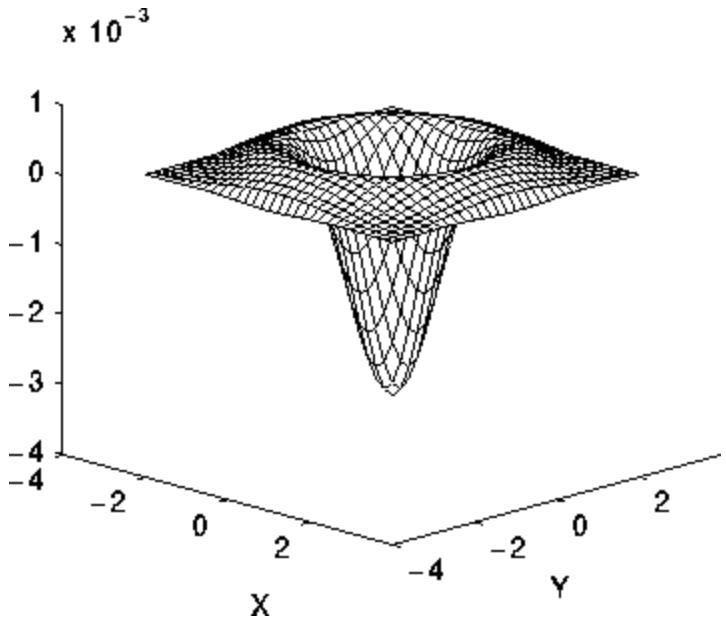
Echivalent cu operatia de convolutie a functiei imagine cu operatorul $\mathbf{LoG}\ (x,y) = \nabla^2 g(x,y)$

Operatorul LoG (2)

LOG: Laplacian of Gaussian $LoG(x,y) = \nabla^2 g(x,y)$

In loc sa se aplice succesiv cei doi operatori de convolutie, se aplica un singur operator, numit "Laplacian of Gaussian", care are expresia:

$$LoG(x,y) = -(1/\pi\sigma^4)*(1 - (x^2 + y^2)/2\sigma^2)*\exp(-(x^2 + y^2)/2\sigma^2)$$



$$LoG(r) = -s * (1 - r^2/2\sigma^2)*\exp(-r^2/2\sigma^2)$$

$$r^2 = (x^2 + y^2), \quad s = 1/(\pi\sigma^4)$$

σ - reprezinta deviatia standard a distributiei;
se alege in functie de nivelul de zgomot din imagine

$$LoG(0) = -s$$

$$LoG(\sigma*2^{1/2}) = 0$$

Operatorul LoG (3)

- Operatorul LoG se aproximeaza in planul discret printr-o masca.
  - Deoarece intereseaza numai punctele de trecere prin zero, in calculul mastii, constanta $1/(\pi\sigma^4)$ se inlocuieste cu un factor de scalare arbitrar, s.
  - LoG(0) = - s
  - ……..
  - $LoG(\sigma*2^{1/2}) = 0$
  - Deci, functia este nenula pe o distanta circulara de $\sigma*2^{1/2}$ de la origine. Latimea filtrului este de $w = 2\sigma*2^{1/2}$ = aproximativ $3*\sigma$.

- Pentru obtinerea efectului dorit, de detectie a pixelilor de front, fara falsi pixeli de front, trebuie sa se utilizeze o masca de marime cuprinsa intre 3w si 4w (pentru reducerea zgomotelor), ceea ce asigura un echilibru intre marimea mastii si performantele operatorului.

- Marimea mastii creste odata cu σ:

σ	$3w = 6\ \sigma\ 2^{1/2}$	Marimea mastii
0.5	4.24	5x5
1	8.48	9x9
2	17	17x17
3	25.4	26x26
4	33.9	34x34
5	42.4	43x43

Operatorul LoG (4)

Urmatoarea masca aproximeaza operatorul LoG pentru σ = 1.4

0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	3	5	4	1
2	5	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	3	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

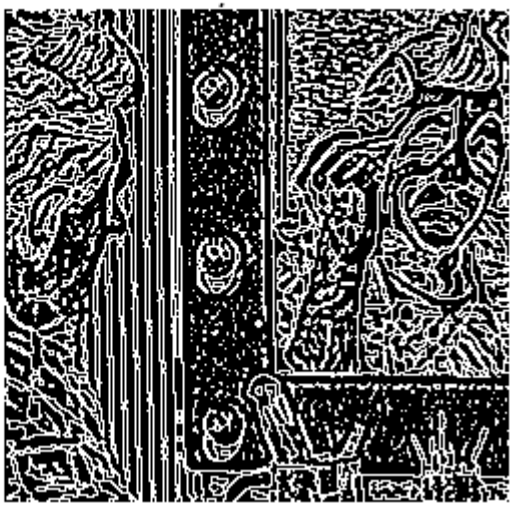
- Pot fi utilizate masti de orice marime dar suma elementelor mastii trebuie sa fie zero (ca si in cazul mastilor Laplace), a.i. rezultatul aplicarii intr-o zona omogena sa fie zero.
- Pentru afisarea iesirii operatorului LoG sub forma unei imagini, valorile din matricea |LOG(x,y)| sunt normalizate la domeniul [0-nivel maxim de intensitate].
- Pentru detectia frontierelor din iesirea operatorului LoG se procedeaza ca si in cazul utilizarii Laplacian-ului.

Operatorul LoG (5)

Imaginea originala

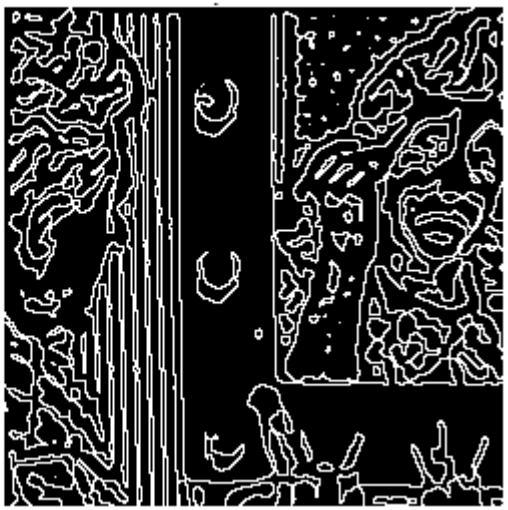
LoG cu deviatia standard 1.0

Matricea frontierelor



LoG cu deviatia standard 2.0

Matricea frontierelor



Evaluarea performantei detectorilor de fronturi (1)

Criterii de evaluare a performantei unui detector de pixeli de frontiera (fronturi) :

- Probabilitatea detectiei de fronturi false
- Probabilitatea de a nu detecta fronturile
- Erori in estimarea directiei fronturilor
- Distanta dintre fronturile detectate si cele reale

Performanţa este influentata de:

- Nivelul de zgomot din imagine
- Marimea mastii filtrului de netezire utilizat inainte de detectie
- Pragul aplicat pe iesirea detectorului pentru a decide daca un pixel este front.

Evaluarea performantei detectorilor de fronturi (2)

Evaluarea performantei :

- Vizuala, examinand imaginea in care sunt afisate fronturile
- Aplicand detectorul pe o imagine sintetica (ex un dreptunghi plin) fara/cu zgomot
- Cantitativa, folosind o masura, de ex. masura Pratt

Pratt's Figure Of Merit (FOM) :

$FOM = (1/\max(Fd, Fi)\ \Sigma\ (1/(1 + a \cdot d_i^2)),\ i=1, Fd$

Fd - numarul de fronturi detectate în imaginea reala

Fi - numarul de fronturi ideale – detectate in imaginea sintetica

d_i – distanta dintre frontul detectat şi cel ideal

a – o constanta utilizata pentru a penaliza fronturile deplasate (1/9 in lucrarea lui Pratt)

- Pentru evaluarea performantei folosind masura Pratt se foloseste o imagine sintetica fara zgomot, in care sunt reprezentate formele din imaginea reala.

Prelucrarea imaginilor color

Prof. univ. dr. ing. Florica Moldoveanu

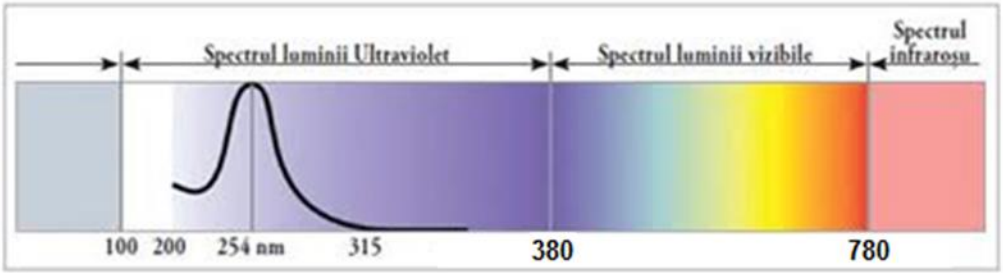
Curs *Sisteme de Prelucrare Grafică* – UPB, Automatică şi Calculatoare
2021-2022

Proprietățile culorilor

- Sistemul vizual uman poate distinge sute de mii de nuanțe de culoare şi intensitate, dar numai în jur de 100 niveluri de gri.

➔ O imagine color furnizează informații suplimentare fața de una în niveluri de gri, care pot fi folosite în analiza imaginilor (identificarea si extragerea obiectelor, bazate pe culoare).

- Culorile percepute de sistemul vizual uman sunt caracterizate prin trei proprietăți:
  - **Nuanța** (o culoare pură), determinată de lungimea de unda dominantă în distribuția spectrală a luminii, perceputa ca o culoare.



- **Saturaţia sau puritatea -** o măsură a amestecului de alb într-o culoare pură.
  O culoare pura are saturatia maxima (100% - nu contine alb).

Proprietățile culorilor

- **Intensitatea**, determinata de energia luminii, **percepută ca luminozitate sau strălucire:**

  Luminozitatea: caracteristică a unui obiect emiţător de lumină

  Strălucirea: caracterizează un obiect neemiţător, care reflectă lumina

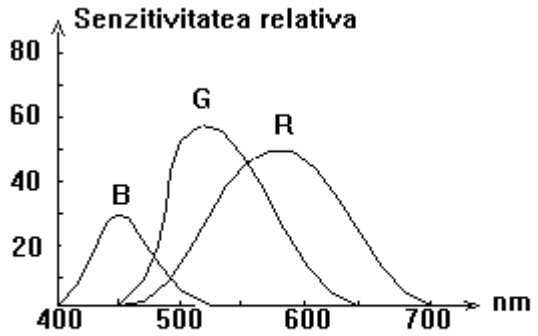
Nuanța + saturația determină cromaticitatea unei culori.

- **Lumina acromatică** nu are culoare. Ea conține în proporții aproximativ egale toate lungimile de undă din spectrul vizibil. **Singura proprietate a sa este intensitatea, măsurată prin nivelul de gri.**

- Lumina perceputa de ochiul uman: emisa (de o sursa) sau reflectata.
- Culoarea luminii reflectate de o suprafață depinde de culoara luminii incidente şi de proprietățile suprafeței (reflectă, absoarbe, transmite lumina).

Teoria tricromaticitatii (Tristimulus theory)
- perceptia culorilor-

- Orice culoare din spectrul vizibil se obtine ca o combinatie intre Rosu, Verde, Albastru.
- Imaginile se formeaza pe retina ochiului uman datorita existentei a trei tipuri de conuri (senzori din retina care contin pigmenti cu absorbtii spectrale diferite):
  - aprox 65% din conuri sunt sensibile la lumina rosie
  - aprox 33% din conuri sunt sensibile la lumina verde
  - aprox 2% din conuri sunt sensibile la lumina albastra



Raspunsul ochiului la cele 3 culori primare
R: 575 nm; G: 535 nm; B: 445 nm

• Culorile roşu (Red), verde(Green) şi albastru (Blue) se numesc culori primare.

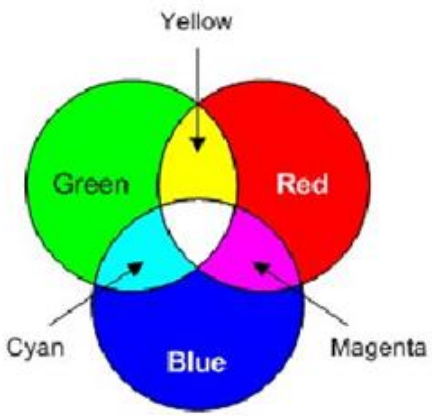
• **RGB: "primitive aditive" -** permit formarea de culori prin **adunarea** lor în diferite proporţii;

Două culori care prin amestec produc lumină albă se numesc complementare.

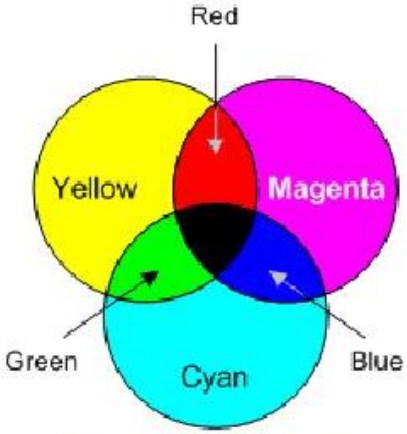
• Culorile complementare culorilor primare sunt : cian (Cyan), magenta (Magenta), galben (Yellow).

• CMY: **"primitive extractive"** - permit formarea de culori prin **extragerea** culorilor primare din lumina alba. Se folosesc pentru a caracteriza culorile reflectate de documentele imprimate.

Teoria tricromaticitatii



Primitive aditive



Primitive extractive

- Definesc lumina perceputa de ochiul uman ca o combinatie de lumina rosie, verde, albastra, în diferite proportii.
- Amestecând două culori primare in proportii egale se obţine culoarea complementară celei de a treia:

  R **G** **B**

  C M Y

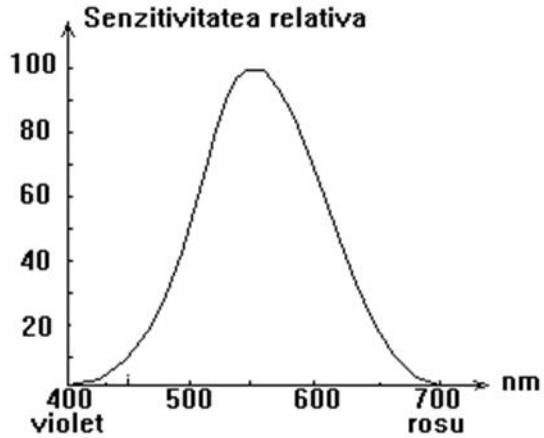
- Definesc culorile reflectate de un document imprimat color.
- Fiecare pigment (primitiva extractiva) extrage din lumina alba una dintre componentele RGB şi reflecta celelalte 2 componente.

  cian = alb-roşu

  magenta = alb-verde

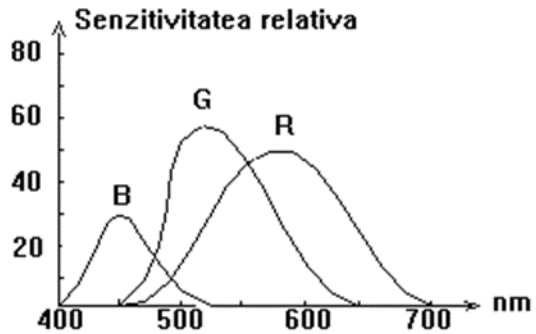
  galben = alb-albastru

Standardul CIE (1)



Curba de luminozitate

Suma celor 3 curbe de senzitivitate



Raspunsul ochiului la cele 3 culori primare

- In 1931, s-a dovedit experimental ca nu orice nuanta care poate fi perceputa de ochiul uman poate fi definita prin adunarea in diferite proportii (pozitive) a culorilor RGB.

- **CIE (*Commission Internationale de l'Éclairage)* a definit 3 primitive (culori primare), denumite *X, Y, Z*, care pot fi combinate prin adunare pentru a forma orice culoare vizibila**.
- (X,Y,Z) se obtin din (R,G,B) printr-o transformare liniara.
- Primitiva *Y* a fost aleasa astfel incat sa se mapeze exact pe curba de luminozitate.

Primitivele CIE (2)

- Primitivele X,Y,Z se obtin din valorile tristimulus (RGB) printr-o transformare liniara, care rezulta din conditiile:

  - orice culoare monocromatica sa fie caracterizata de valori tristimulus pozitive
  - Y sa fie o masura a eficientei luminoase (luminozitatii culorii)
  - pentru lumina alba valorile tristimulus sa fie egale

$$\begin{vmatrix} X \\ Y \\ Z \end{vmatrix} = \begin{vmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{vmatrix} * \begin{vmatrix} R \\ G \\ B \end{vmatrix}$$

- Coeficientii tricromatici x, y, z – contributiile relative ale celor trei primitive X,Y,Z in formarea unei culori.

$$x = X/(X+Y+Z)$$
$$y = Y/(X+Y+Z)$$ Coeficientii tricromatici
$$z = Z/(X+Y+Z)$$
$$x+y+z = 1$$

❖ Orice culoare din spectrul vizibil este definita prin:
- cromaticitate – o pereche (x,y), deoarece z= (1-x-y)
- intensitate – primitiva Y

Diagrama de cromaticitate

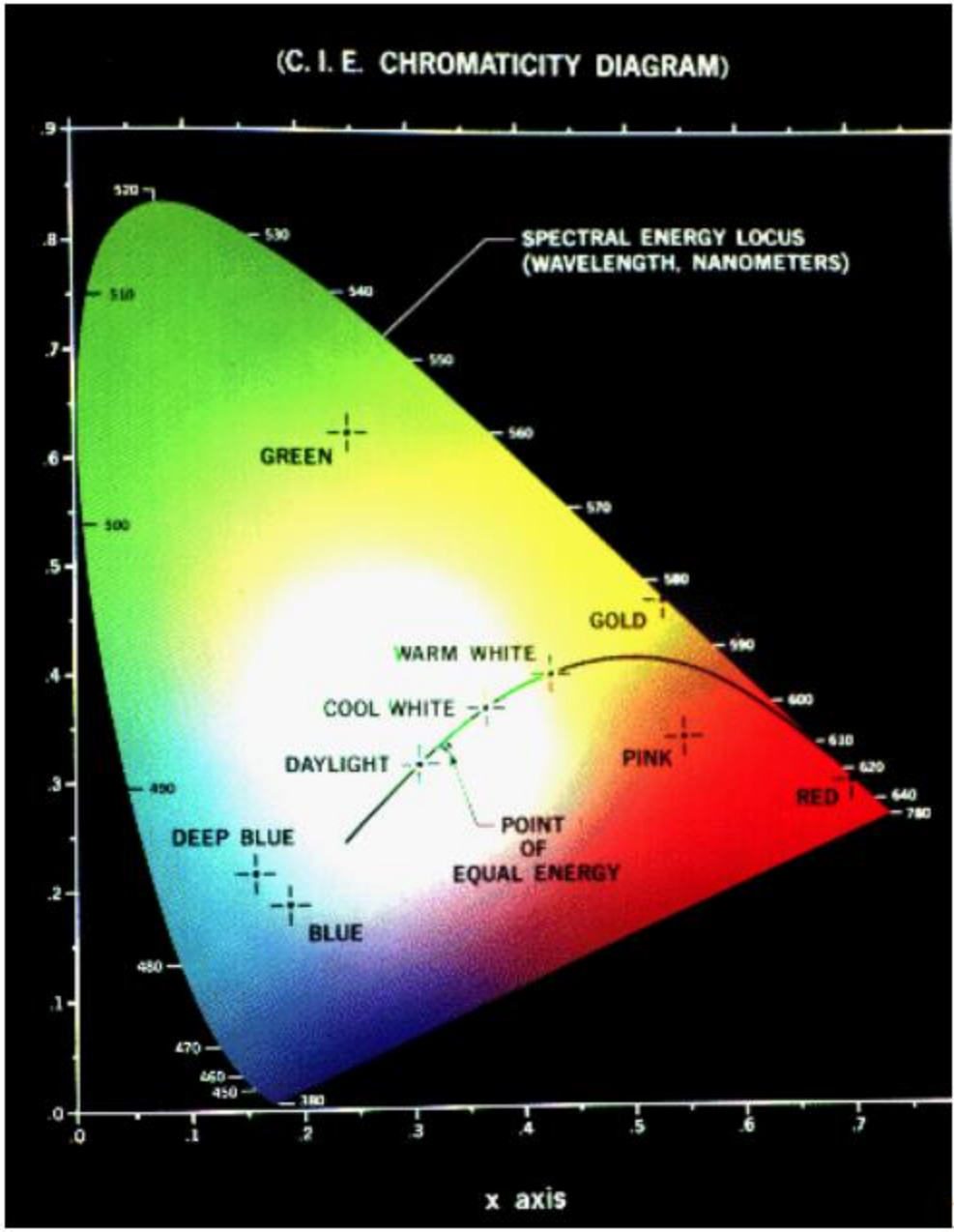
- Diagrama de cromaticitate ilustreaza compunerea culorilor ca functii de x si y.

- **Culorile pure** (nuante) sunt reprezentate pe marginile diagramei: (380 – 780) nm.

- **Saturatia:**
- maxima (100%) pe marginile diagramei
- Descreste spre interior
- 0 in punctul de egala energie

- Cromaticitatea = nuanță + saturație. Este independentă de energia luminoasă.

- **Culorile cu aceeaşi cromaticitate, dar diferite intensitați sunt reprezentate prin acelaşi punct în diagrama de cromaticitate.**



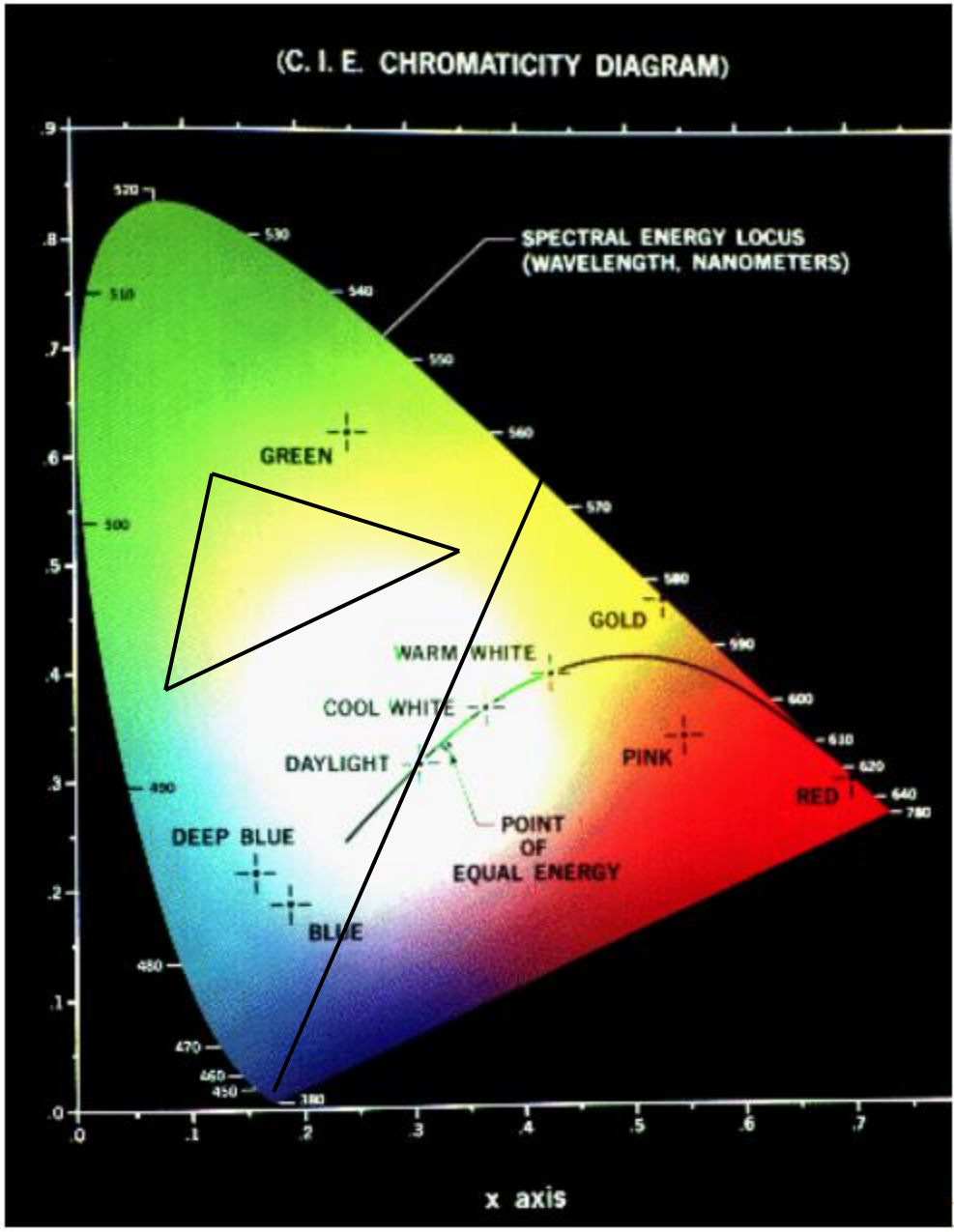
Imagine din “Gonzalez&Woods, Digital Image Processing (2002)”

Diagrama de cromaticitate

- “Point of equal energy”: punctul in care cele trei culori primare au proportii egale (x=y=z=1/3)
  →culoarea alb in standardul CIE.

- Culorile complementare se află în extremitatile unei linii care trece prin acest punct.
- Toate culorile de-a lungul oricărei linii din diagrama cromaticității pot fi obținute prin amestecarea culorilor din extremitatile liniei.
- Toate culorile dintr-un triunghi pot fi formate prin amestecarea culorilor din vârfuri.

- Nu toate culorile vizibile pot fi reprezentate prin amestec de R,G,B (sau alte 3 culori primare) deoarece diagrama nu este triunghiulara.



Modele de culoare(1)

Se folosesc mai multe **modele de culoare**:

- **Orientate catre utilizator** – bazate pe proprietatile psiho-fiziologice ale culorilor (nuanta, luminozitatea, saturatia): HSV, HLS, HSI

- Orientate catre echipamente şi transmisia imaginilor

- Reflectă modul de formare a culorilor la echipamentele de vizualizare şi la cele de imprimare: RGB, CMY, YIQ, YCbCr.

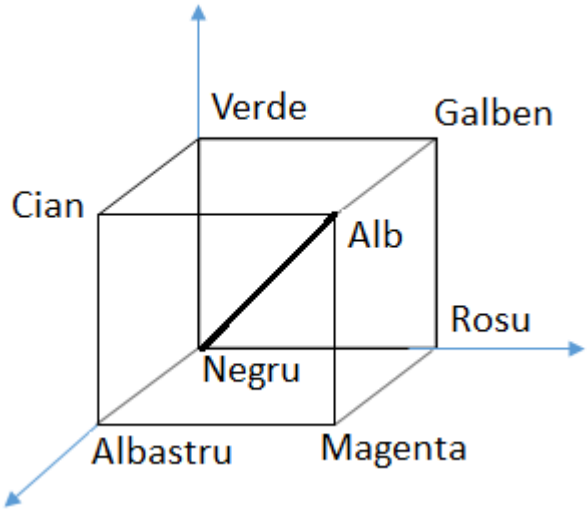
Modele de culoare utilizate frecvent in prelucrarea imaginilor: HSI, HSV, RGB, YIQ, YCbCr

- Un <u>model de culoare</u> specifică un sistem de coordonate 3D şi un subspaţiu al culorilor în sistemul de coordonate respectiv.
- Fiecare culoare se reprezintă printr-un punct în subspaţiul culorilor.

Modelul RGB - (Red, Green, Blue)

Defineste culorile la dispozitivele emitatoare de lumina

Culoare emisa: C(x,y,z) = x_Red + y_Green + z_Blue, 0<= x,y,z <=1



Amestecul aditiv de R,G,B pentru formarea culorilor complementare.

Rosu: 1,0,0

Verde: 0,1,0

Albastru: 0,0,1

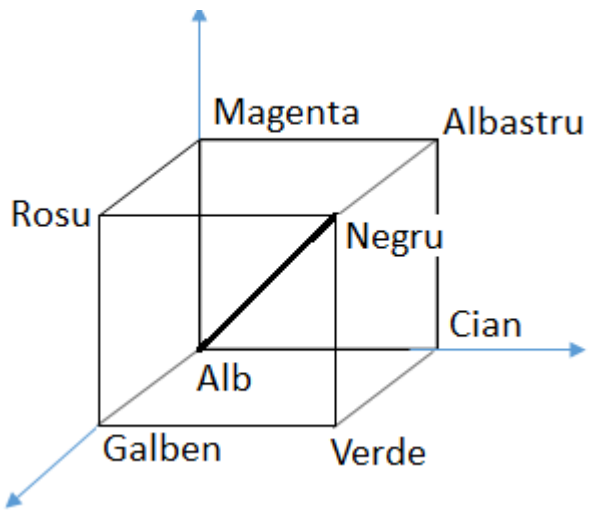
Alb: 1,1,1; Negru: 0,0,0

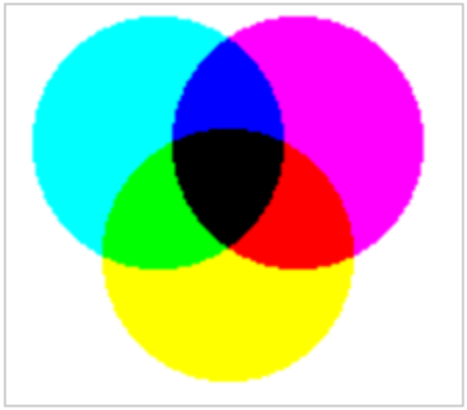
Cian: 0, 1, 1 - culoarea complementara culorii Rosu – amestec de verde cu albastru
Galben: 1, 1, 0 - culoarea complementara culorii Albastru - amestec de rosu cu verde

Modelul CMY (Cyan, Magenta, Yellow)

Defineste culorile reflectate de documentele imprimate

Culoare reflectata: C(x,y,z) = (1,1,1) – (x,y,z) =
(1-x) + (1-y) + (1-z), 0<= x,y,z <=1





Amestecul extractiv de C,M,Y pentru formarea culorilor reflectate (în RGB)

Coversia RGB ←→ CMY

[C M Y] = [1 1 1] - [R G B]

[R G B] = [1 1 1] - [C M Y]

Cian= (1,0,0) → (1,1,1)– (1,0,0) = (0,1,1) in RGB (culoarea reflectata în RGB)

Galben = (0,0,1) → (1,1,1) - (0,0,1) = (1,1,0) in RGB

Alb = (0,0,0) →(1,1,1) – (0,0,0) = (1,1,1) in RGB

Negru = (1,1,1) → (1,1,1) – (1,1,1): (R,G,B) sunt extrase in proportie maxima din lumina alba

Modelul CMYK (Cyan, Magenta, Yellow, Black)

- Combinatia CMY = (1,1,1) nu produce negru perfect.
- Majoritatea imprimantelor folosesc culoarea negru → componenta K
- K se foloseste pentru a define o gama mai larga de culori: definirea unei culori in CMYK este dependenta de dispozitiv.
- Conversia din CMY in CMYK (c,m,y,k): mai multe metode.

Exemple de conversii:

1) K = min (C,M,Y)

daca (K==1)

atunci (c,m,y,k) =(0, 0, 0, 1)

altfel

(c,m,y,k) = ((C-K)/(1-K), (M-K)/(1-K), (Y-K)/(1-K), K

2) (C, M, Y) → (C-K, M-K, Y-K, min (C,M,Y))

Modelul YIQ (luminance-inphase-quadrature)

Modelul YIQ - Utilizat in transmisia imaginilor TV color, in sistemul NTSC.

- Y: luminanţa – intensitatea luminii
- I,Q – cromaticitatea
- Proiectat pentru a mentine compatibilitatea cu televiziunea alb-negru: componenta Y contine toata informatia pentru TV monocrom.

$$\begin{vmatrix} Y \\ I \\ Q \end{vmatrix} = \begin{vmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{vmatrix} * \begin{vmatrix} R \\ G \\ B \end{vmatrix}$$

(Efectuata inainte de transmisia imaginilor TV)

$$\begin{vmatrix} R \\ G \\ B \end{vmatrix} = \begin{vmatrix} 1 & 0.956 & 0.621 \\ 1 & -0.272 & -0.647 \\ 1 & -1.105 & 1.702 \end{vmatrix} * \begin{vmatrix} Y \\ I \\ Q \end{vmatrix}$$

(Efectuata la receptia imaginilor)

Avantajul modelului in procesarea imaginilor: decupleaza luminanța de culoare → **se poate procesa componenta de luminanță fara a afecta componentele de cromaticitate.**

Modelul YCbCr

Se foloseste in special in transmisia imaginilor color comprimate.

- Canalul Y codifica luminanta (intensitatea) – reprezentare perfecta a imaginii in nivele de gri

- Canalele Cb si Cr reprezinta cromaticitatea – contin mai putina informatie si sunt mai usor de comprimat

Conversia RGB – YCbCr (la compresie):

$Y = 0.299R + 0.587G + 0.11B$

$Cb = -0.169R -0.331G + 0.5B$

$Cr = 0.5R – 0.419G – 0.081B$

Conversia YCbCR – RGB (la decompresie)

$R = Y + 1.402Cr$

$G = Y – 0.344Cb – 0.714Cr$

$B = Y + 1.772Cb$

Canalul Verde are cea mai mare pondere în Y: ochiul uman are senzitivitatea maxima la verde.

Imaginea YCbCr

Canalul Y

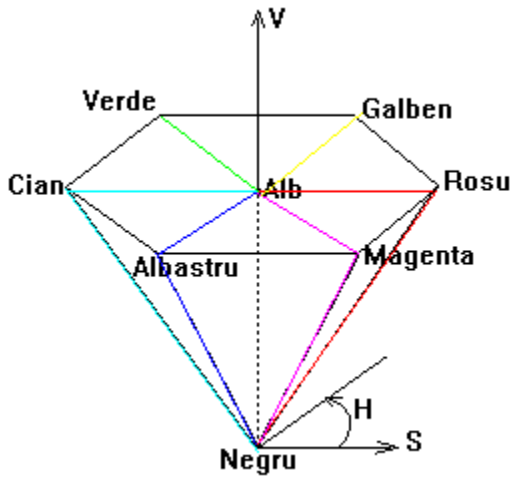
Canalul Cb

Canalul Cr

Modelul HSV (Hue, Saturation, Value)

- Model de culoare orientat catre utilizator
- Artiştii specifică culorile prin tente, nuanţe şi tonuri

Hexaconul din figura reprezinta subspatiul culorilor in modelul HSV.



0<= V<= 1, 0<= S <= 1, 0<= H <= 360

V=1 – baza hexaconului

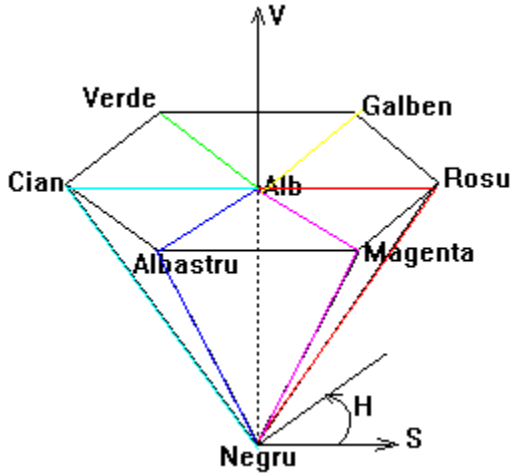
S=0 - pe axa hexaconului

H=0 - culoarea Rosu

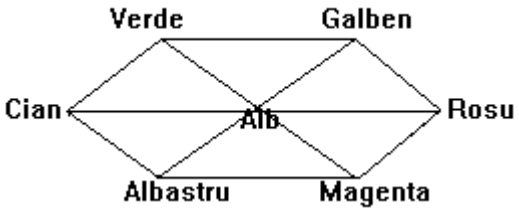
Culorile complementare – la 180 grade pe baza hexaconului

Avantajul modelului in prelucrarea imaginilor: separa intensitatea de componentele de cromaticitate.

Modelul HSV (Hue, Saturation, Value)



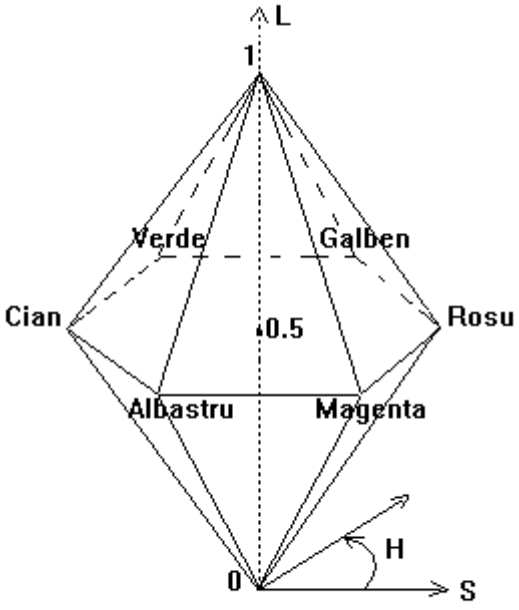
- Saturaţia este definită prin distanţa de la axa hexaconului la marginile hexaconului, pentru toate valorile lui V.
- Combinaţia (S=0,V=1) corespunde culorii alb iar prin (S=0, 0<=V<=1) se reprezintă nivelurile de gri.
- Pentru S=0 valoarea lui H este nesemnificativă.
- Culorile pure sunt reprezentate prin V=1 şi S=1.



- Baza hexaconului corespunde suprafeţei văzute atunci când se priveşte cubul RGB dinspre vârful (1,1,1) de-a lungul diagonalei principale.

- Fiecare plan de V constant din spaţiul HSV corespunde vederii unui subcub din cubul RGB.
- Diagonala principală a cubului RGB devine axa principală în spaţiul HSV.
- Această interpretare sta la baza algoritmilor de conversie între modelele RGB şi HSV.

Modelul HLS (Hue, Lightness, Saturation)



Modelul HLS

- Introdus de Tektronix. Denumit si HSL.
- Culorile primare şi complementarele lor sunt reprezentate prin S=1, L=0.5.

Conversii RGB → HSV/HLS

max = max(R, G, B) min = min(R, G, B)

V = max

S= (max-min)/max daca max ≠ 0

0 daca max =0

H = (G – B) / (max – min) daca R = max

(B – R) / (max – min) + 2 daca G = max

(R – G) / (max – min) + 4 daca B = max

daca H<0 H = H +6

H = H x 60

RGB → HSV

RGB: 125,200,100

→HSV=105,0.5,0.78

→HSL=105,0.476,0.588

L = (max-min)/2

S = (max – min)/(max + min) daca L<=0.5

= (max – min)/ (2 – max – min) daca L > 0.5

H = (G – B) / (max – min) daca R = max

(B – R) / (max – min) + 2 daca G = max

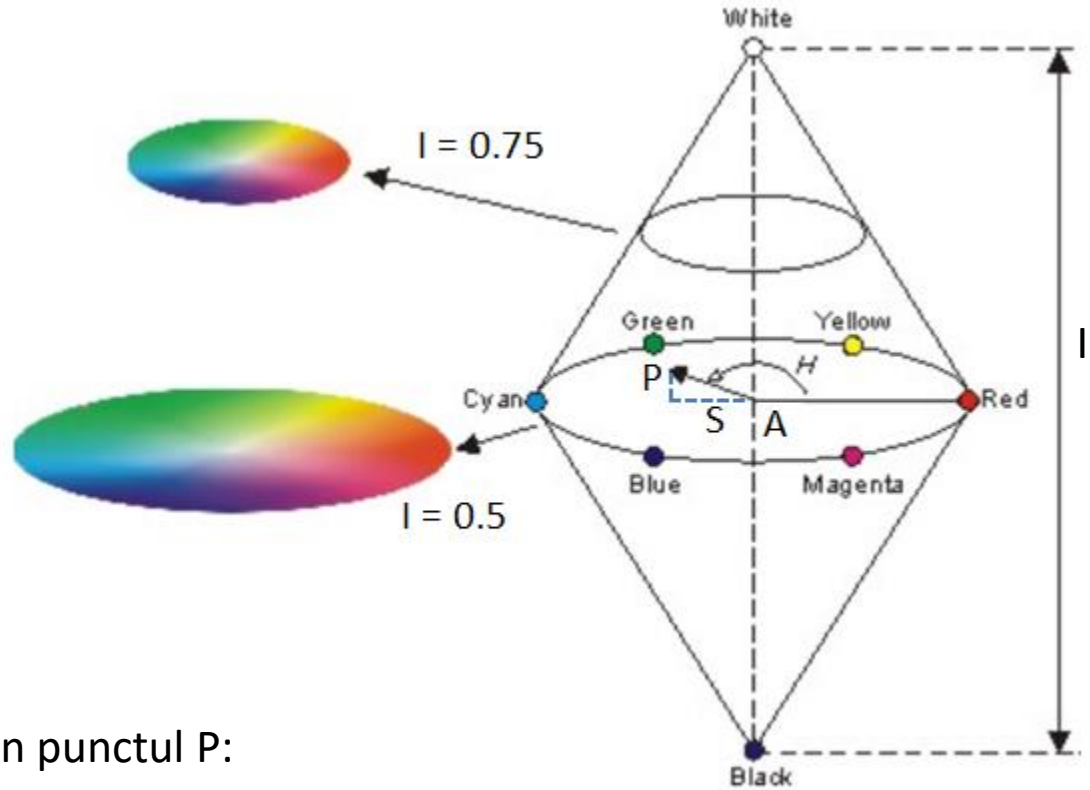
(R – G) / (max – min) + 4 daca B = max

daca H<0 H = H +6

H = H x 60

RGB → HLS

Modelul HSI (Hue, Saturation, Intensity)



H=0: Rosu
H = 120: Verde
H = 240: Albastru
H = 60: Galben
H = 300: Magenta
➢ La fel ca in HSV

- Fie P un punct in spatiul HSI: P(H,S,I)
- AP – vectorul din punctul de pe axa verticala in planul I la P

In punctul P:

- Intensitatea (I) este data de coordonata verticala (variaza de la 0 la 1 pe axa verticala)
- Saturatia (S): distanta lui P fata de axa verticala (S=0 pe axa, S= 1 pe frontiera volumului)
- Nuanta (H): unghiul pe care-l face vectorul AP cu axa Red (Red: H=0)

Avantajul modelului:

- Separa intensitatea de cromaticitate: → se pot face prelucrari separat pe cromaticitate si intensitate.

Conversie RGB → HSI

Pentru 0<= R,G,B<=1

Se obtin componentele HSI:

$$H = \cos^{-1}\left\{\frac{0.5\cdot\left[(R-G)+(R-B)\right]}{\left[(R-G)^2+(R-B)(G-B)\right]^{1/2}}\right\} \qquad H \in [0,180] \quad \text{pentru} \quad B \le G$$

$$H = 360 - \cos^{-1}\left\{\frac{0.5\cdot\left[(R-G)+(R-B)\right]}{\left[(R-G)^2+(R-B)(G-B)\right]^{1/2}}\right\} \qquad H \in [181,360] \quad \text{pentru} \quad B > G$$

$$S = 1 - 3 \cdot \min(R,G,B)/(R+G+B) \qquad S \in [0,1]$$

$$I = (R+G+B)/3 \qquad I \in [0,1]$$

R,G,B:255,255,125

→HSV = 60,0.5,1

→HSL=60,1.0,0.745

→HSI=60,0.4,0.83

https://www.picturetopeople.org/p2p/image_utilities.p2p/color_converter?color_space=RGB&color_channel1=255&color_channel2=255&color_channel3=125&ResultType=view

Tehnici de prelucrare a imaginilor color(1)

- Doua categorii de metode de prelucrare a imaginilor color:

  - Se proceseaza separat fiecare componenta a culorii si se compune rezultatul; in unele cazuri prelucrarea se aplica numai uneia dintre componente, de exemplu, intensitatea sau nuanta;

  - Se lucreaza cu vectori de culoare (fiecare culoare are trei componente, fiind reprezentata printr-un vector in spatiul culorilor). De exemplu, culoarea unui pixel p(x,y) se reprezinta in spatiul (R,G,B) prin vectorul (0,0,0) →(Cr(x,y), Cg(x,y), Cb(x,y)).

- Modelul de culoare folosit intr-o aplicatie de prelucrare/ analiza de imagini depinde de scopul aplicatiei.

Transformari de culoare (1)

Fie f(x,y) imaginea de intrare, g(x,y) imaginea de iesire si o transformare de intensitate, T:

g(x,y) = T(f(x,y)) sau

s = T(r), unde r este o intensitate in imaginea de intrare iar

s este intensitatea in imaginea de iesire

O transformare de culoare este:

(s1, s2, s3) = T(r1, r2, r3), unde ri şi si sunt componentele celor 2 culori

Transformarea se poate aplica numai uneia dintre componentele culorii.

Imbunatatirea contrastului: se aplica numai componentei intensitate

In HSV, HSI: s3 = T(r3); in HLS: s2 = T(r2)

Evidentierea culorilor de interes:

- poate fi efectuata in RGB, prin definirea unui subvolum al culorilor de interes
- poate fi efectuata in HSV, HLS, HSI, definind subintervalul de interes din H (0-360)

Transformari de culoare (2)

Evidentierea culorilor de interes in spatiul RGB

- Subspatiul culorilor de interes dintr-o imagine poate fi reprezentat printr-un cub sau printr-o sfera:
  - Cubul cu latura w si centrul in (a1,a2,a3)
  - Sfera de raza R si centrul in (a1,a2,a3)

  (a1,a2,a3) poate fi media culorilor din subspatiul culorilor de interes.

- Pixelii a caror culoare nu se incadreaza in volumul de interes sunt setati la o culoare "neutrala", pe care o notam cu n, ceilalti isi pastreaza culoarea.

- Atunci, transformarea si = T (ri) este:

1. si = n, daca (|rj – aj| > w/2), 1<= j <=3
   ri, altfel
2. si = n, daca $(r1-a1)^2 + (r2 – a2)^2 + (r3 – a3)^2 > R^2$
   ri, altfel

Transformari de culoare (3)

Complementul unei imagini RGB

- Fiecare culoare are o culoare complementara asa dupa cum un nivel de gri are un negativ:

  Negativul unui nivel de gri, g: Neg(g) = Lmax – g.

- Culorile complementare culorilor R, G, B sunt, respectiv, C(Cyan), M(Magenta) si Y(Yellow).
- Calculul complementului unei imagini color inseamna inlocuirea fiecarei culori din imagine cu complementul său:

  C = 1- R, M = 1-G, Y = 1- B.

  Transformarea este: (s1,s2,s3) = (1-r1, 1-r2, 1-r3)

- Operatia este importanta in imbunatatirea detaliilor din zonele intunecate ale unei imagini color.

Histograma unei imagini color

- Se calculeaza in special in spatiile de culoare care separa componentele de cromaticitate de componenta intensitate: YIQ, YCbCR, HSV, HLS, HSI
  - histograma culorilor din imagine, folosind componenta “hue”
  - histograma saturatiei
  - histograma luminantei: componenta Y/V/L/I
- Pentru imagini RGB histograma se construieste separat pentru fiecare canal de culoare: pt. analiza statistica, s.a.

Egalizarea histogramei

Poate fi efectuata la fel ca pentru imagini cu nivele de gri:

asupra componentei care reprezinta luminozitatea (I din HSI, Y din YIQ, V din HSV, L din HLS), lasand neschimbate celelalte componente.

Filtrarea imaginilor color

- In cazul filtrelor spatiale liniare, definite prin diferite masti de convolutie, filtrarea se aplica:

  - Separat asupra componentelor culorii, in RGB, considerandu-se ca fiecare canal poate fi afectat de zgomot; exemplu: aplicarea filtrului medie pe o fereastra de 3x3 pixeli:

$$g_\lambda(x,y) = 1/9 \ \Sigma \Sigma \ f_\lambda(x+i, y+j) , -1 <= i, j <= 1$$

λ = R/G/B , f_λ – componentele Red, Green, Blue ale imaginii

  - Asupra componentei de intensitate (ex. I in HSI sau Y in YIQ): variatii mari ale intensitatii in pixeli vecini pot indica prezenta zgomotului.

Detecţia pixelilor de front din imagini color (1)

- Folosind modelele HSV, HLS, HSI, YCbCR, se pot detecta pixelii de front pe baza:

  1. Diferenţelor în intensitate, folosind componentele V, L, I, Y

  2. Diferentelor cromatice, folosind componenta H/ componente care definesc cromaticitatea

  Detecţia se efectueaza la fel ca pentru imagini cu mai multe nivele de gri.

- Folosind modelul (R,G,B), se pot detecta pixelii de front pe baza:

  1. Diferentelor de intensitate

     Se converteste imaginea color in imagine cu mai multe nivele de gri:

     nivel-gri = Y (din modelul YIQ) = (0.11 * B + 0.59 * G + 0.3 * R)

     Se aplica un detector de fronturi pe imaginea in nivele de gri

  2. Diferenţelor cromatice

     2.a. Efectuand detectia separat pentru cele 3 canale de culoare

     2.b. Calculand gradientul vectorului culoare

Detecţia pixelilor de front din imagini color (1)

2.a. – detectia fronturilor separat pentru cele 3 canale de culoare – 2 metode

- Se calculeaza amplitudinile gradientilor pe cele 3 canale: f_R, f_G, f_B

$$Dr(x,y) = Grad(f_R(x,y)),\ Dg\ (x,y)= Grad(f_G(x,y)),\ Db\ (x,y)= Grad(f_B(x,y))$$

Se calculeaza $F_R(x,y)$, $F_G(x,y)$, $F_B(x,y)$ – matricile fronturilor din Dr, Dg, Db (de ex. aplicand un prag pe Dr, Dg, Db)

Iesirea detectorului de fronturi pt imaginea color se aproximeaza prin:

$$F(x,y) = F_R(x,y) + F_G(x,y) + F_B(x,y)$$

Este metoda cea mai populara

- Se calculeaza $Dr(x,y) = Grad(f_R(x,y)),\ Dg\ (x,y)= Grad(f_G(x,y)),\ Db\ (x,y)= Grad(f_B(x,y))$

Se calculeaza $D(x,y) = Dr\ (x,y) + Dg(x,y) + Db(x,y)$

Se calculeaza F(x,y) = matricea fronturilor din D(x,y)

Detecţia pixelilor de front din imagini color (1)

2.b. Culorile sunt considerate vectori.

Se calculeaza gradientul imaginii in care fiecare pixel este un vector culoare.

- O imagine in nivele de gri este reprezentata printr-o functie scalara, f(x,y). Gradientul său intr-un punct (x,y) este un vector a carui directie indica rata maxima de schimbare a functiei f in punctul (x,y).
- Culoarea este un vector intr-un spatiu tri-dimensional. O definitie mai exacta a gradientului unei imagini color se bazeaza pe gradientul (marime si directie) vectorului culoare (*).

(*) Andreas Koschan and Mongi Abidi, Detection and Classification of Edges in Color Images, A review of vector-valued techniques, IEEE SIGNAL PROCESSING MAGAZINE[64], JANUARY 2005

Extragerea frontierelor din imagini

Prof. univ. dr. ing. Florica Moldoveanu

Extragerea frontierelor (1)

Extragerea frontierelor pe baza punctelor de frontiera

- Detectia pixelilor de frontiera
- Eliminarea falselor puncte de frontiera
- Conectarea punctelor de frontiera validate pentru a forma frontiere.

Se folosesc:

Metode locale, care utilizeaza relatiile bazate pe intensitate ale fiecarui pixel din imagine cu pixelii vecini.

Metode globale, care se bazeaza pe cunoasterea formei geometrice a frontierelor sau a ecuatiei matematice (ex. Transformata Hough).

Binarizarea (Thresholding) matricei amplitudinilor

- Extragerea frontierelor → binarizarea matricei amplitudinii gradientului, prin aplicarea unui prag.
- In imaginea binara: pixelii =1 reprezinta fronturi, restul au valoarea zero

Globala:

$$B(k,l) = \begin{cases} 1 & daca \quad a(k,l) \geq T \\ 0 & altfel \end{cases}$$

a(k , l) este matricea amplitudinilor
T este valoarea de prag, aceeasi pentru toti pixelii imaginii

T se poate alege pe baza histogramei matricei amplitudinilor, a.i. numai un mic procent din elementele matricei sa aiba valoarea peste prag

Alegerea pragului: mare →intreruperi in frontiera; mica→frontiere late

Adaptiva:

$$B(k,l) = \begin{cases} 1 & daca \quad a(k,l) \geq T(k,l) \\ 0 & altfel \end{cases}$$

T(k,l) este pragul adaptat local

T(k,l) = med(k,l)(1 + p)*

med(k,l) este media aritmetica locala, calculata intr-o vecinatate a pixelului

p: pragul peste media locala

Detectorul Canny(1)

A fost proiectat cu urmatoarele obiective:

- scaderea ratei de eroare – detectorul sa nu piarda puncte de frontiera si sa nu raspunda la puncte care nu sunt puncte de frontiera
- punctele de frontiera sa fie bine localizate – distanta dintre punctele de frontiera gasite de detector si cele reale sa fie minima
- detectorul sa furnizeze un singur raspuns la un singur punct de frontiera

Paşii:

- Reducere zgomote – netezire imagine
- Detectie puncte de front – calcul gradient imagine: amplitudine + directie front in fiecare pixel
- Localizarea frontierelor si eliminarea falselor puncte de frontiera, folosind matricea amplitudinilor si cea a directiei fronturilor

Detectorul Canny(2)

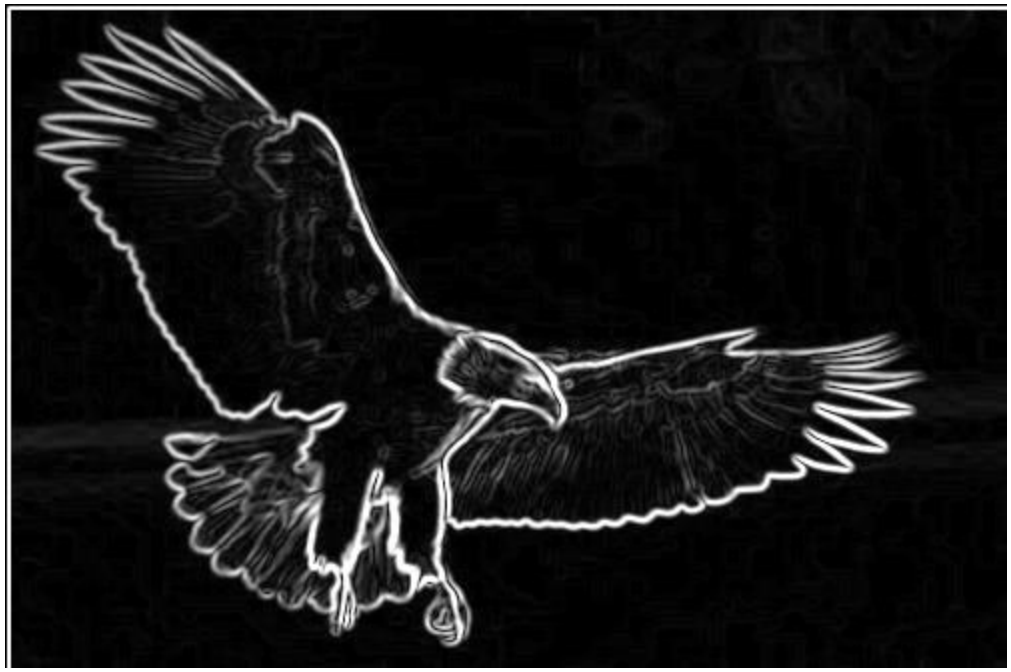
Imaginea originala

Pasul 1:

•Se aplica imaginii de intrare un filtru Gaussian.

•Cu cat latimea mastii este mai mare cu atat este mai mica senzitivitatea detectorului la zgomot. Totodata, eroarea de localizare a fronturilor creste odata cu latimea mastii Gaussiene.

Detectorul Canny(3)



Pasul 2:

• Se aplica operatorul Sobel imaginii rezultate din pasul 1.

2.1. Se calculeaza matricea amplitudinii gradientului.

$$D(x,y) = |Dx| + |Dy|$$

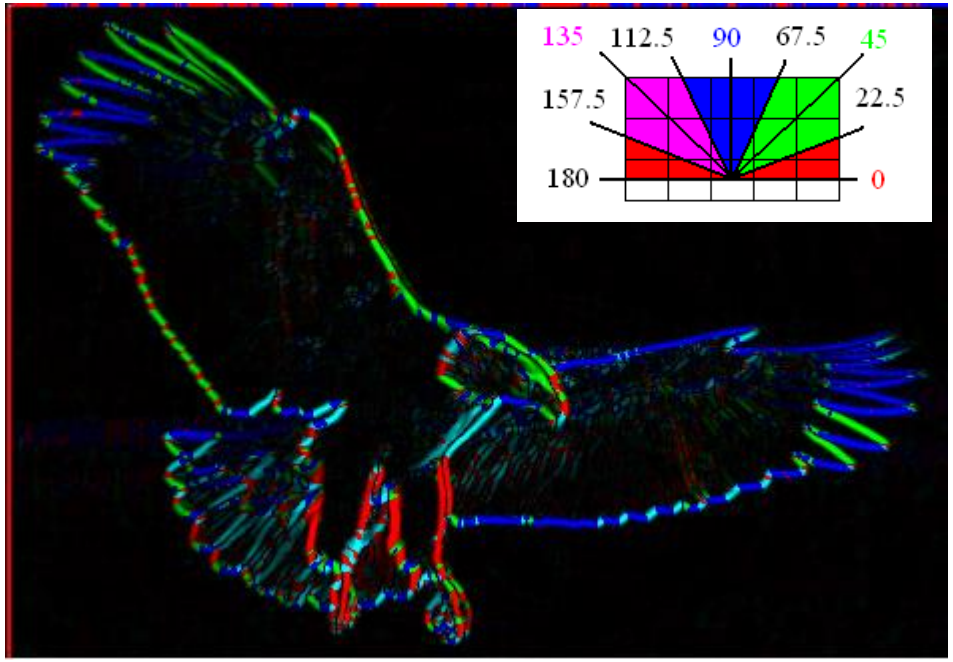
2.2. Se calculeaza directia gradientului in fiecare pixel, obtinandu-se matricea directiilor:

$\theta(x,y) = \mathrm{arctg}(Dy / Dx)$, pentru $Dx \# 0$

0, pentru $Dy=0$

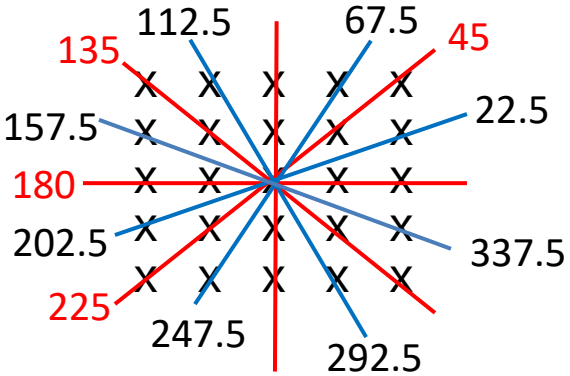
+/- 90, daca $Dx=0$, $Dy \# 0$

Detectorul Canny(4)



2. 3.

Se ajusteaza θ(x,y) la una dintre directiile spatiului discret al imaginii, si anume la cea mai apropiata de valoarea sa.



Directia ajustata a gradientului:

Orizontala: θ←0 daca: -22.5 <= θ < 22.5 sau -157.5 <= θ <= 157.5

La 45 grade: θ←45 daca: 22.5<= θ < 67.5 sau 202.5<= θ < 247.5

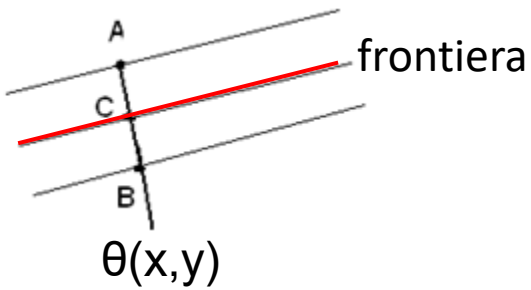
Verticala : θ←90 daca: 67.5<= θ < 112.5 sau 247.5<= θ < 292.5

La 135 grade: θ←135 daca: 112.5<= θ < 157.5 sau 292.5<= θ < 337.5

Detectorul Canny(5)

Pasul 3 (non-maxima suppression): subțierea frontierelor

- In matricea amplitutidinilor, frontierele apar ca zone late
- Se subtiaza frontierele eliminand pixelii care nu au amplitudinea gradientului maxima locala.



D – matricea amplitudinilor
C = (x,y) – pixelul curent
A şi B – pixelii de pe directia ajustata a gradientului, θ(x,y)

Daca D(A) > D(C) sau D(B) > D(C)

atunci D(C) ← 0

altfel D(A) ← 0 şi D(B)← 0

➢ Sunt declarate puncte de frontiera acele puncte care au amplitudinea gradientului maxima locala pe directia gradientului. Celelalte puncte sunt eliminate (setate la zero).

Efectul: subtierea frontierei fara intreruperea ei.

Detectorul Canny(6)

Matricea amplitudinilor

Iesirea pasului 3
(subtierea frontierelor)

Detectorul Canny(7)

Pasul 4 (hysteresis thresholding) – Extragerea frontierelor

Pasul 1 nu a eliminat complet zgomotele - acestea se manifesta ca pixeli cu intensitate redusa in apropierea frontierei

Se elimina pixelii falsi de frontiera (zgomotele) fara intreruperea frontierei:

- **Se utilizeaza operatia de "histerezis", care consta in aplicarea a 2 praguri:**
  - pragul de jos, P1, si pragul de sus, P2 (P2 aproximativ egal cu 2*P1).
  - aceste praguri se aplica separat matricei amplitudinilor rezultata in pasul anterior, rezultand 2 imagini binare, T1 si T2:
    - In T1 au valoarea 1 pixelii cu amplitudinea >P1 iar in T2 aceia cu amplitudinea >P2.
    - Imaginea din T2 are intreruperi in frontiera dar contine mai putine fronturi false.

Detectorul Canny(8)

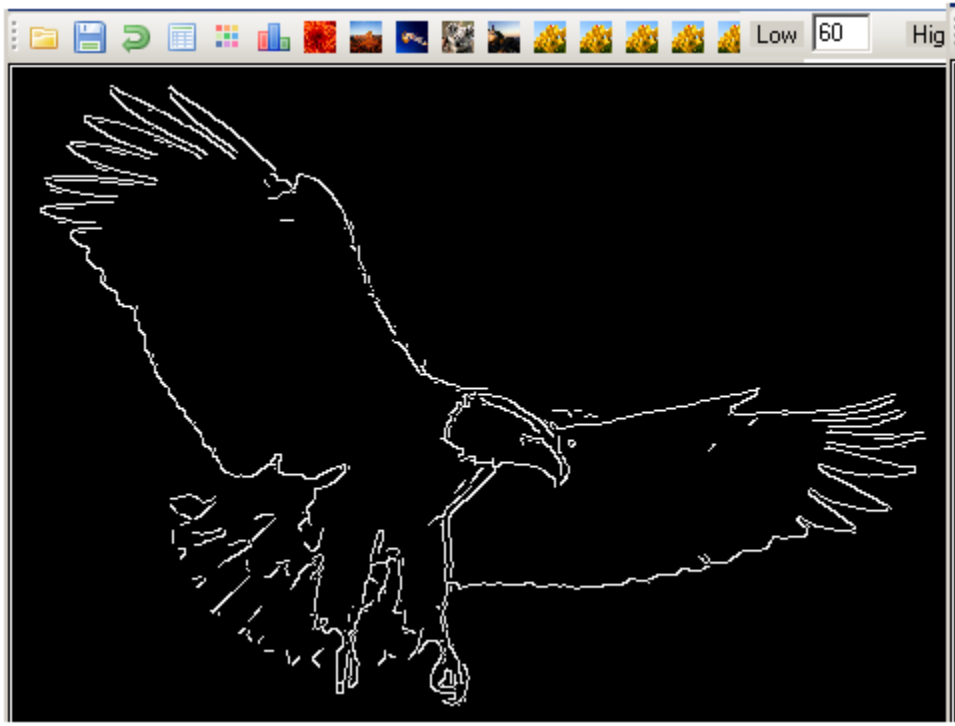
Pasul 4 (hysteresis thresholding) - cont

- Iesirea pasului 4: - frontierele, reprezentate printr-o matrice binara/lista adreselor pixelilor de frontiera
- Frontierele se obtin legand punctele din T2 si folosind puncte din T1 acolo unde apar întreruperi în T2:
  - Se incepe cu un punct din T2.
  - Se conecteaza puncte din T2 (marcandu-le in matricea de iesire/adaugandu-le la lista pixelilor de frontiera) pana cand se ajunge intr-un pixel p care nu mai poate fi conectat cu alt punct din T2 (nu are vecini in T2 in vecinatatea sa de 8 pixeli).
  - Se cauta in T1, in vecinatatea de 8 pixeli a pixelului p, un pixel care poate fi conectat la frontiera
  - Se conecteaza la frontiera puncte din T1 pana cand se ajunge la un pixel cu valoare diferita de zero in T2 (sau se ajunge la sfarsit de frontiera)

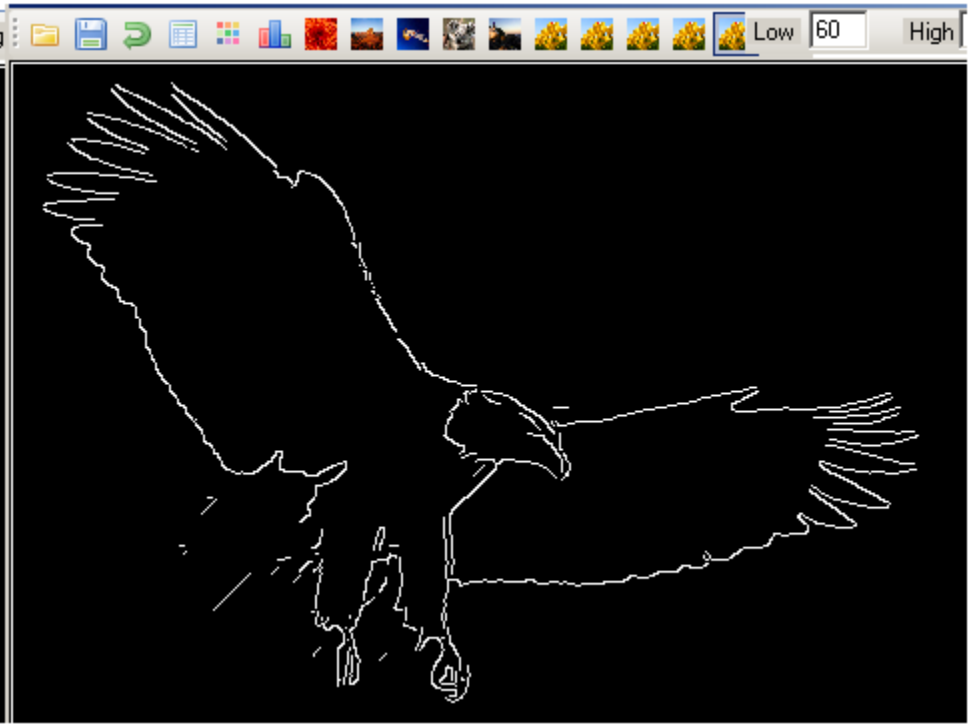
Detectorul Canny(9)

•Metoda histerezis rezolva problemele care apar la utilizarea unui singur prag, dar rezultatul depinde de alegerea pragurilor P1 si P2.

Pasul 4- efectul alegerii pragurilor P1 si P2



P1 = 60, P2 = 130



P1 = 60, P2 = 180

Urmarirea frontierei prin predictia punctului urmator(1)

a(P) = a(x,y) amplitudinea frontului in pixelul P(x,y)

θ(P) = θ(x,y) directia frontului in pixelul P(x,y)

T, T1 si T2 – valori de prag, date.

- **Două puncte de front, P_i, P_j , pot fi conectate dacă diferenţele în amplitudine şi direcţie sunt sub valorile de prag date iar amplitudinile fronturilor sunt relativ mari.**
    1. $| a(P_i) – a(P_j) | <= T1$
    2. $| \theta (P_i) - \theta (P_j) | <= T2 \bmod 2\pi$
    3. $a(P_i) >=T$ si $a(P_j) >=T$
- Se considera vecinatati mici (de regula 3x3 pixeli): $\mathbf{P_i}$ si $\mathbf{P_j}$ se cauta in vecinatatea considerata.

Urmarirea frontierei prin predictia punctului urmator(1)

Algoritmul constă in :

- Determinarea punctului de start
- Determinarea celorlalte puncte de frontiera
  - Din punctul curent:
    - Se aproximeaza pozitia urmatorului punct: punctul de frontiera anticipat
    - Se alege următorul punct de frontiera din vecinatatea celui anticipat

Determinarea punctului de start

Se cauta doua puncte, P1, P2, dintr-o vecinatate de 3x3 pixeli, care satisfac conditiile 1,2,3:

- cunoscand zona sau directia frontului
- parcurgand matricea amplitudinilor si a directiilor

Predicţia punctului următor (punctul anticipat)

- Se bazează pe punctele de frontieră deja determinate şi eventual pe cunoaşterea tipului de frontieră (linie dreapta, curba neteda).
- **Punctul anticipat** se afla la o distanta **d** de punctul curent, pe frontiera determinata pana la momentul curent.

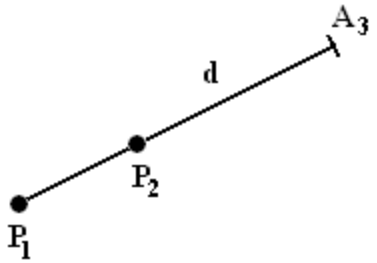
Urmarirea frontierei prin predictia punctului urmator(1)

Determinarea punctului de frontiera următor

- Punctul următor este căutat în vecinătatea punctului anticipat.
- El trebuie să fie un punct de front care poate fi conectat cu ultimul punct de frontieră determinat, conform condiţiilor (1) + (2) + (3).

Fie P_1 şi P_2 primele 2 puncte de frontiera, alese conform condiţiilor (1) + (2) + (3).

- Punctul anticipat, A_3, se află la o distanţă **d** de P_2 pe dreapta determinată de $P_1 – P_2$

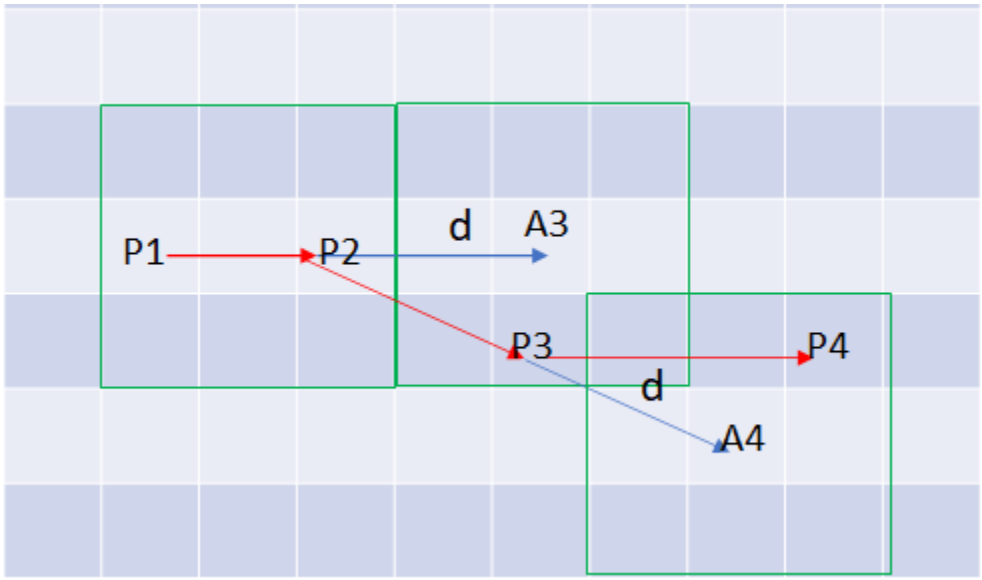


A_3 poate să nu fie adresă de pixel sau poate să nu fie un pixel de front.

Urmarirea frontierei prin predictia punctului urmator(1)

- Punctul de frontieră următor, P_3, se caută în vecinătatea lui A_3, astfel încât să fie un punct de front care împreună cu P_2 să satisfacă conditiile (1) + (2) + (3):

În continuare, direcţia de căutare este dată de vectorul P2-P3.



• Dacă distanţa **d** este mare, procesul de urmărire a frontierei este puţin sensibil la zgomot, adecvat pentru curbe netede.

➢ Pentru curbe cu zone de curbură mare (fluctuaţii mari), **d** trebuie să fie mic.

Extragerea liniilor drepte din imagini
Transformata Hough

Prof. univ. dr. ing. Florica Moldoveanu

Detectia liniilor drepte (1)

❖ Metoda globala de extragere a frontierelor: se bazeaza pe ecuatia matematica a frontierelor

Intrarea: B - matricea obţinută prin binarizarea matricei amplitudinilor fronturilor.

Iesirea: o matrice binara in care pixelii cu valoarea 1 apartin unor linii drepte.

Imaginea: N1xN2 pixeli

➢ **Determinarea liniilor drepte folosind perechi de pixeli:**

Numărul maxim de linii posibile in imagine este $N(N-1)/2$, unde $N = N_1 \times N_2$

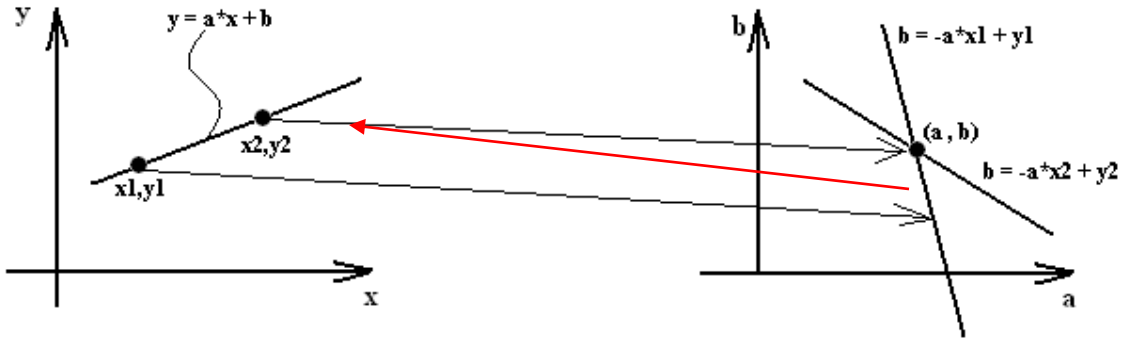
În cazul cel mai defavorabil, fiecare pixel este verificat dacă aparţine uneia dintre cele $N(N-1)/2$ linii:

- complexitatea calculului este $O(N^3)$

Transformata Hough (1962, Paul Hough) **utilizează o reprezentare parametrică a liniilor** pentru reducerea complexităţii calculului căutării lor în imagini:

$$y = ax + b$$

Detectia liniilor drepte (2)



- În spaţiul parametric (a , b) fiecare linie dreapta din spatiul (x,y) este reprezentată printr-un singur punct.
- Intersectia liniilor corespunzatoare punctelor (x1,y1) si (x2,y2) in spatiul parametric determina in mod unic dreapta (x1,y1) – (x2,y2)

Daca dreapta y = a*x + b exista in imagine

→ exista cel putin 2 pixeli care satisfac ecuatia

→ exista cel putin doua drepte in spatiul parametric care se intersecteaza in punctul (a,b).

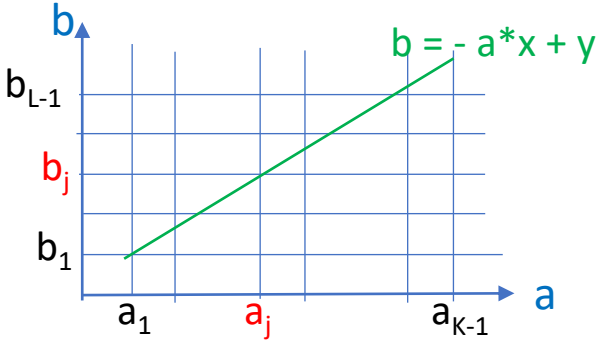
Numarul de drepte care se intersecteaza in (a,b) = numarul de pixeli din imagine, care se afla pe dreapta y=a*x + b

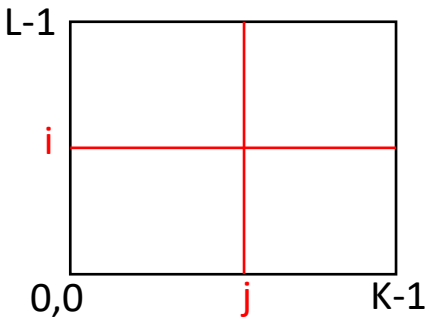
Detectia liniilor drepte (3)

1. Se discretizează spaţiul parametric şi se reprezintă printr-o matrice P (numita *acumulator*), astfel încât fiecare P[i][j] corespunde unui punct al spatiului discretizat:

$a_1 <= a <= a_{K-1}$, $a_j = a_1 + j*da$

$b_1 <= b <= b_{L-1}$, $b_i = b_1 + i*db$





Se inițializează elementele matricei P cu zero.

2. Pentru fiecare pixel (x, y), cu B[y][x] = 1 (pixel de front)

// se formeză ecuaţia *b = - a*x + y* (dreapta coresp. pixelului in spatiul parametric)

for (j=0; j<K; j++) // pentru fiecare valoare discreta a lui a

{ $a_j=a_1 + j*$ da

$b_j = - a_j*x + y$ // (a_j, b_j) punct de pe dreapta b = -a*x + y

$i = (b_j – b_1)/(b_{L-1} – b_1) * (L-1)$ // linia din matricea P corespunzatoare punctului

P[i][j] ++; // numara liniile din spatiul parametric discretizat care trec prin (a_j, b_j)

}

Detectia liniilor drepte (4)

3. Pentru fiecare element P[i][j] al matricei P:

dacă P[i][j] >= T (prag) atunci

{ // in imagine exista dreapta y = a*x + b, unde a = a_1 + j* da, b = b_1 + i*db

//marcheaza in matricea de iesire pixelii care se afla pe dreapta y = a*x + b

pentru fiecare pixel p(xp,yp) din B:

daca B[yp][xp) =1 && yp = a*xp + b

atunci out[yp][xp] = 1

- **Precizia metodei** depinde de pasul de discretizare a spaţiului (a , b), deci de mărimea matricei P.
- **<u>Complexitatea calculului transformatei Hough este O(K*N)</u>**, unde ***K*** este numărul de subintervale pe axa ***a*** iar N este numărul de pixeli din imaginea de intrare.

pentru fiecare pixel (x, y) din matricea de intrare

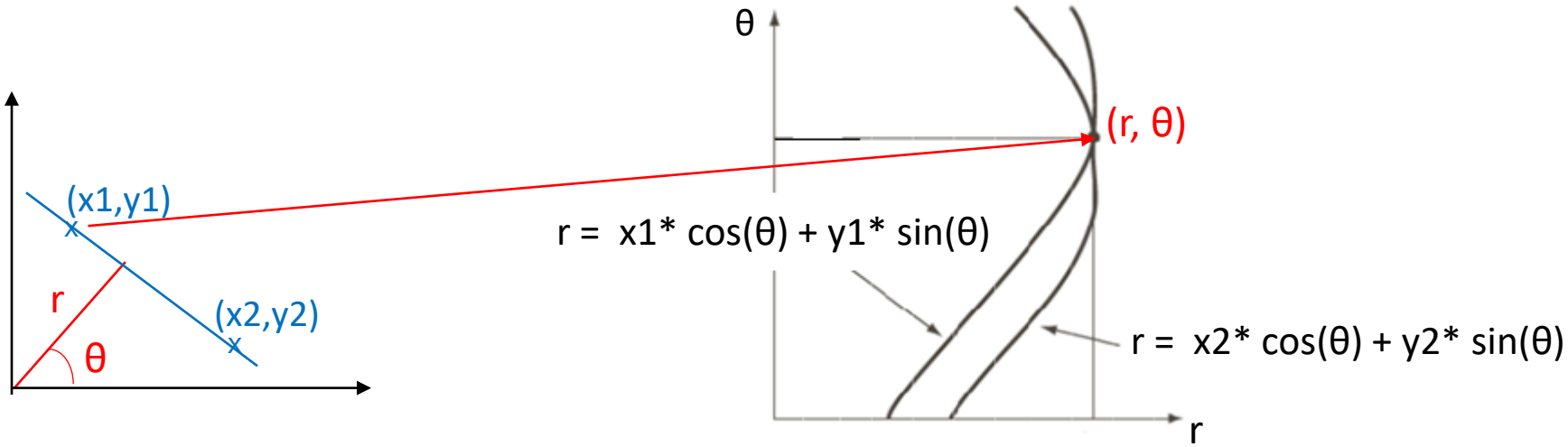
for (j=0; j<K; j++) //pt fiecare valoare discreta a lui a

❖ Utilizarea ecuaţiei parametrice y = a*x + b produce dificultăţi în reprezentarea liniilor verticale (a-> ∞).

Detectia liniilor drepte (5)

O reprezentare parametrica mai convenabilă este reprezentarea polară :

r = x* cos(θ) + y* sin(θ) unde r este distanta de la origine la dreapta.



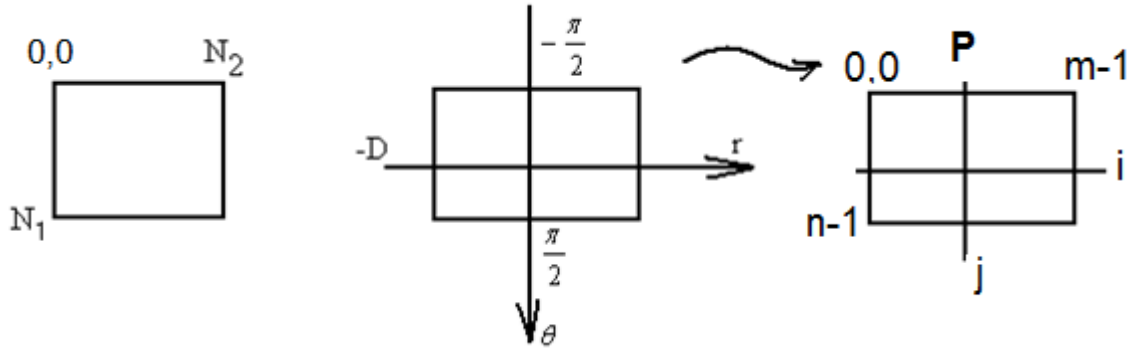
- Un punct (x_1, y_1) se reprezintă printr-o curbă sinusoidală în spaţiul parametric (r, θ):

  r = x1* cos(θ) + y1* sin(θ)

➢ Puncte colineare, (x_i, y_i), din imagine, corespund intersecţiilor sinusoidelor din spaţiul parametric.

Implementare (1)

Fie o imagine de N_1xN_2 pixeli si D lungimea diagonalei imaginii, $D = \sqrt{N_1^2 + N_2^2}$



$r = -D,\ j = 0$

$r = D,\ j = m - 1$

$\theta = -\frac{\pi}{2}, i = 0$

$\theta = \frac{\pi}{2}; i = n - 1$

Discretizarea spaţiului parametric :

- intervalul $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ se împarte în *n* subintervale;

- intervalul [*-D, D*] se împarte in *m* subintervale;

- P[n][m] matricea corespunzatoare punctelor grilei spaţiului parametric.

Coloana j pentru o val a lui r: *j*=(int)(0.5 + (r+D)(m-1) /(2D))

Unghiul coresp liniei i din P: $\theta = (\frac{i * \pi}{n - 1} - \frac{\pi}{2})$

Implementare (2)

```
void Hough (imagine B,//matricea binară a amplitudinilor
            int n, int m, int N1, int N2, int T, imagine out)
{
  int P[N][M], float COS[N] , float SIN[N]; // N si M sunt constante
```

//Se prelcalculeaza valorile funcţiilor *sin* şi *cos* pentru valorile discrete ale lui θ

```
    float t;
    for ( i=0; i<n; i++)
    {
      t = ( i*M_PI/(n-1) - M_PI_ 2 );
      COS[i] = cos(t);
      SIN[i] = sin(t);
    }
```

$$\theta = \left(\frac{i * \pi}{n-1} - \frac{\pi}{2}\right)$$

Implementare (3)

```
  float D = sqrt((float) N1*N1 + (float) N2*N2);
//Se initializeaza matricea P
  for (k = 0; k < n; k ++)
   for(l = 0; l < m; l ++)
         P[k][l] = 0;

// Se numara sinusoidele care trec prin punctele discrete ale spatiului (θ,r)
 for (k = 0;  k < N1; k ++)
   for(l = 0;  l < N2;  l ++)
   { if (B[k][l] == 1) //pixel de frontiera
        for(i = 0; i < n; i++) //pentru fiecare θ
           {r = l*COS[i] + k*SIN[i]; // calc. r din ec. curbei → sinusoida coresp lui B[k][l]
            j = (r + D)*(m - 1)/(2*D) + 0.5; // calc. coloana din P
            P[i][j] ++;
           }
    }
```

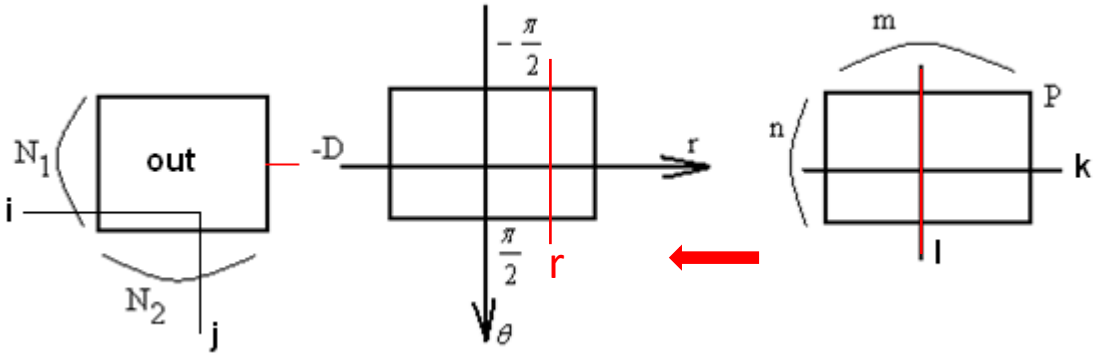
Implementare (4)

// Se calculează transformata Hough inversă (din spaţiul (r , θ) în (x , y))

- **Pentru fiecare element P[k][l] > T, se înscrie în matricea out, sub formă de pixeli = 1, linia dreaptă corespunzătoare elementului (k , l)**

// Se efectuează operaţia <u>and</u> între rezultatul transformatei inverse şi matricea de intrare

```
for(k = 0 ; k < n ; k++)
   for(l = 0 ; l < m ; l++)
    { if ( P[k][l] > T )
     { r = (2.0 * D * l)/(m-1) – D; //val lui r corespunzatoare coloanei l: (r+D)/2D = l/(m-1)
       // se marcheaza in matricea out pixelii care satisfac ecuatia   r = x*COS[k] + y*SIN[k]
      for(j= 0 ; j < N2 ; j++)
       {  if ( SIN[k] == 0 )  y = N1/2; //θ = 0
          else  y = ( r – j * COS[k])/SIN[k]);
          i = y + 0.5;
         if ( i >= 0 && i < N1 && B[i][j] == 1)
            out[i][j] = 1;
       }
      }
     }
} //Hough
```



Implementare (5)

- **Complexitatea calculelor transformatei Hough poate fi redusa daca se dispune de matricea directiei fronturilor , Φ(x , y).**

Diferenţiind în raport cu x in ambele părţi ale ecuaţiei :

r = x* cos(θ) + y* sin(θ)

rezulta:

$$\frac{dr}{dx} = \cos(\theta) + \frac{dy}{dx} * \sin(\theta) = 0 \quad \Rightarrow \quad \frac{dy}{dx} = -\frac{\cos(\theta)}{\sin(\theta)} = -\cot(\theta) = tg(\theta + \frac{\pi}{2})$$

Direcţia frontului in (x , y) este: $\phi_{x,y} = arctg(\frac{dy}{dx})$ = Φ(x , y)

Deci: $\frac{dy}{dx}$ = tg(Φ(x , y)) =$tg(\theta + \frac{\pi}{2})$ → Pentru un pixel (x,y):

θ = Φ(x , y) - π/2

Transformata Hough (11)

Rezultă că pentru fiecare pixel B[k][l] = 1, numai un singur element al matricei P trebuie să fie actualizat : $P[i_\theta][j_\theta]$

$\theta = FI[k][l] - \pi/2$ // unde FI este matricea directiei frontului

$i_\theta = (\theta/\pi + 0.5)(n-1)$

$r_\theta = l^* \cos[i_\theta] + k^* \sin[i_\theta]$

$j_\theta = (int)\ r_\theta$

$P[i_\theta][j_\theta]$ ++

in locul ciclului →

```
if (B[k][l] == 1) //punct de frontiera
  for(i = 0; i < n; i++) //pentru fiecare θ
   {r = l*COS[i] + k*SIN[i]; // calc. r din ec. curbei
    j = (r + D)*(m - 1)/(2*D) + 0.5; // calc. col in P
    P[i][j] ++;
   }
```

Complexitatea calculului transformatei Hough se reduce in acest caz de la O(n*N) la O(N) → N=numarul de pixeli din imaginea de intrare

- Transformata Hough se poate generaliza pentru a detecta orice curbă parametrică.
- Metoda este practică numai pentru curbe cu un număr mic de parametri: de exemplu, pentru un cerc cu ecuatia $(x-a)^2 + (y-b)^2 = r^2$, parametrii sunt r, a , b. Este necesară o matrice P tridimensională, P(r , a , b).

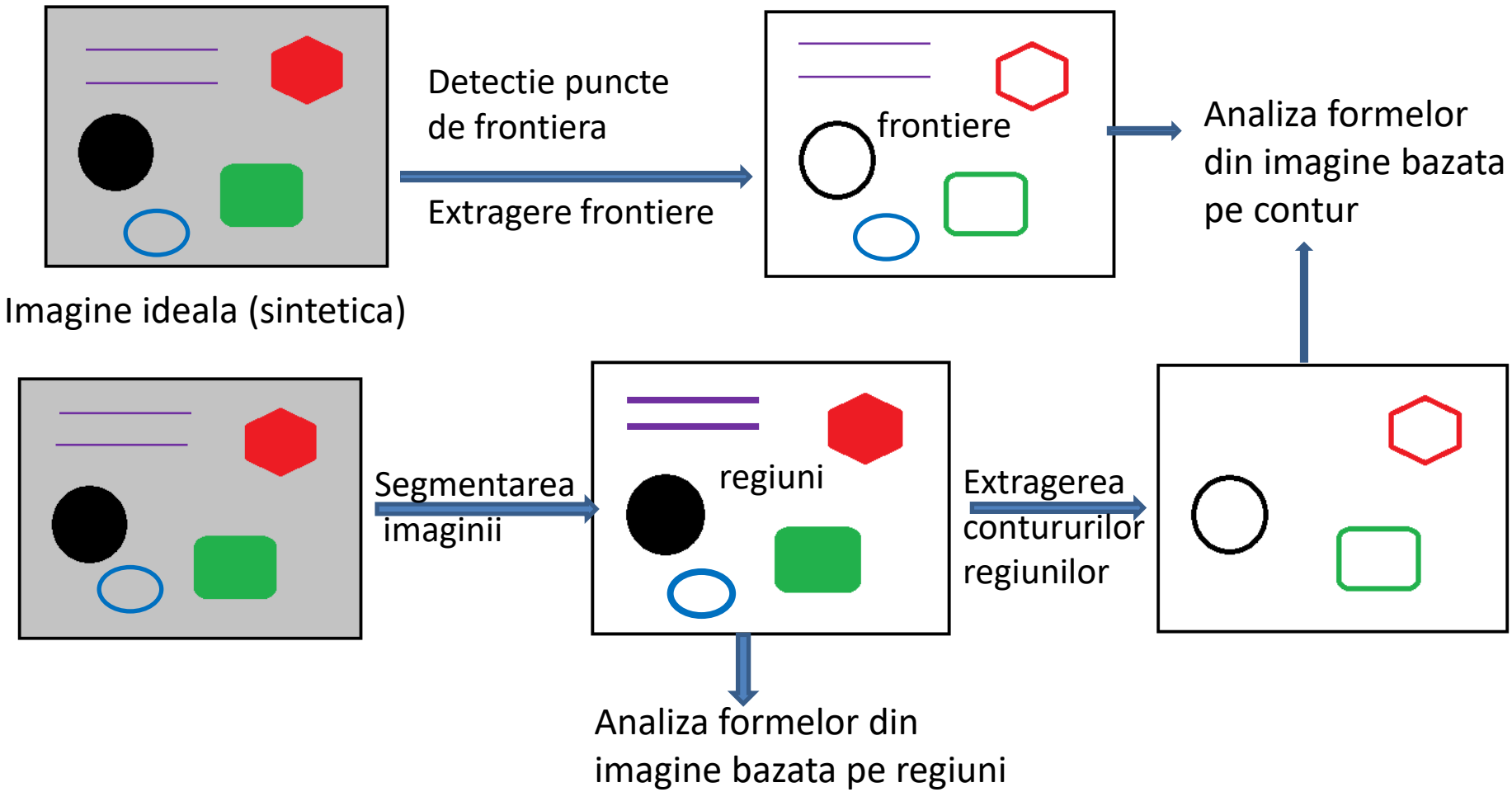
Segmentarea imaginilor

Prof. univ. dr. ing. Florica Moldoveanu

Curs *Sisteme de Prelucrare Grafică* – UPB, Automatică şi Calculatoare
2021-2022

Segmentarea

❖ Determinarea zonelor (regiunilor) dintr-o imagine care conţin pixeli cu anumite caracteristici similare: intensitate, culoare, adâncime – într-o imagine RGBD (D: Depth).



Metode de segmentare

I . **Segmentarea bazata pe determinarea frontierelor regiunilor** (detectia fronturilor, unirea punctelor de front pentru delimitarea regiunilor).

II. **Segmentarea bazata pe regiuni**

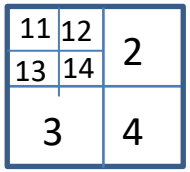
- II.1. Segmentarea prin divizare („ splitting ”)
- II.2. Segmentarea prin divizare şi unificare („ split & merge ”)
- II.3. Segmentarea prin extindere („region growing”)
- II.4. Segmentarea prin extindere şi unificare („region growing and merging”)

III. **Segmentarea bazata pe praguri de intensitate (la nivel de pixel).**

IV. **Segmentarea prin clusterizarea pixelilor pe baza intensitatilor (la nivel de pixel)**

Segmentare prin divizare recursiva(1)

- Se verifică dacă întreaga imagine este omogenă, conform unui criteriu. Dacă nu este omogenă, atunci imaginea este divizată în patru regiuni egale. Se verifica omogenitatea fiecărei regiuni. Procedura de divizare este aplicată recursiv până când se obţin numai regiuni omogene.




```
int omogen( imagine a, int N1, int M1, int N2, int M2, int T )
{  int max =0, min = Lmax;
   for(int i=M1; i<M2; i++)
        for(int j=N1; j< N2; j++)
                { if(a[i][j] < min) min = a[i][j];
                  if(a[i][j] > max) max = a[i][j];
                }
   if(max – min < T) return 1; // criteriul de omogenitate
   else return 0;
}
```

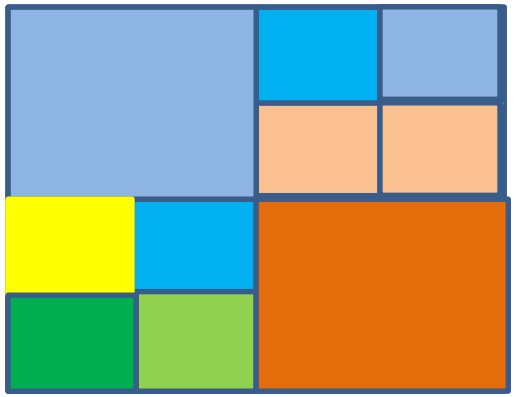
Functia omogen verifica omogenitatea unei regiuni dreptunghiulare (N1, M1) – (N2,M2)

Segmentare prin divizare recursiva(2)

```
void divideReg(imagine a, imagine b, int N1, int M1, int N2, int M2, int T, int *N)
{ // a este imaginea de intrare iar b este imaginea de iesire
 // (N1, M1) – (N2,M2) sunt colturile stanga –sus si dreapta-jos ale regiunii segmentate
// T este pragul folosit in criteriul de omogenitate
// in N se memoreaza numarul de regiuni rezultate din divizarea recursiva
long s;
if(!omogen(a,N1, M1, N2, M2,T) && (N2-N1) > 1 && (M2-M1) >1)
{ // se divide regiunea in 4 zone egale si se apeleaza functia pentru cele 4 zone
  divideReg(a, b, N1, M1, (N1+N2)/2, (M1+M2)/2, T, N);
  divideReg(a, b, (N1+N2)/2, M1, N2, (M1+M2)/2, T, N);
  divideReg(a, b, N1, (M1+M2)/2, (N1+N2)/2, M2 , T, N);
  divideReg(a, b, (N1+N2)/2, (M1+M2)/2, N2, M2, T, N);
}
else
// regiune omogena; calculeaza media intensitatilor pixelilor regiunii si o asigneaza
// pixelilor regiunii in imaginea de iesire
{ (*N)++;
  s=0;
  for(int i=M1; i<M2; i++)
        for(int j =N1; j<N2; j++)
                s+ = a[i][j];
  s/ =(N2-N1)*(M2-M1);
  for(i=M1; i<M2; i++)
        for(j=N1; j<N2; j++)
                b[i][j] = (unsigned char) s;
}
return;
}
```

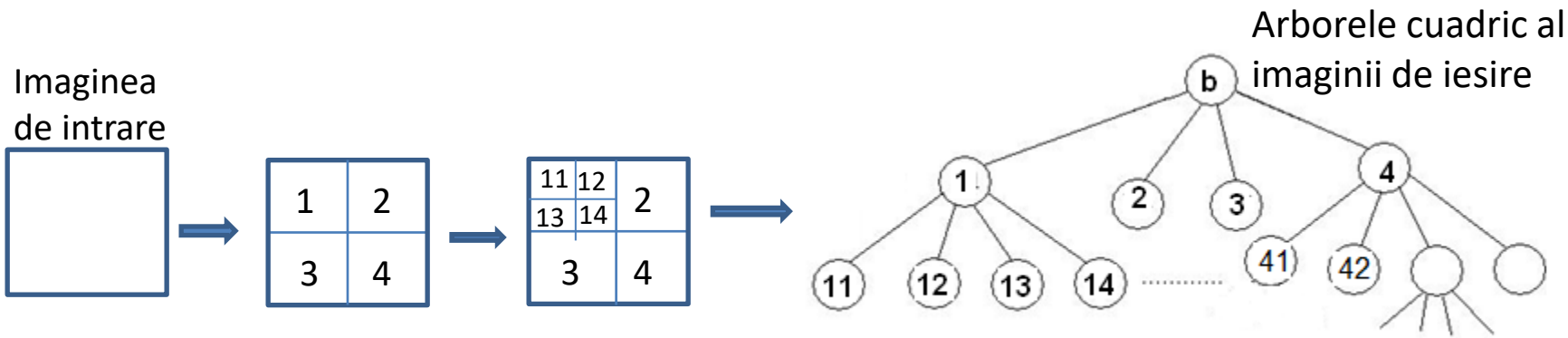
1	2
3	4

Imaginea de iesire

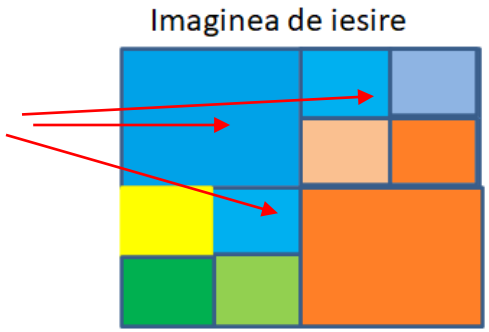


Segmentare prin divizare recursiva(3)

Algoritmul de segmentare prin divizare recursivă produce o imagine segmentată care poate fi reprezentată printr-un arbore cuadric :



- Se memoreaza în fiecare nod al arborelui intensitatea care a fost atribuită pixelilor regiunii.
- **Dezavantajele algoritmului:**
  - **pot rezulta regiuni adiacente similare**
  - **poate produce o suprasegmentare**



- Solutia: dupa segmentare sunt unificate regiunile adiacente similare folosind arborele cuadric al imaginii segmentate.
- Unificarea a 2 regiuni: se calculeaza media intensitatilor regiunilor si se atribuie pixelilor regiunilor unificate.

Segmentarea prin divizare şi unificare (1)

Fie P predicatul care defineşte regula de segmentare (de exemplu Imax – Imin < prag)

dacă Ri si Rj sunt adiacente si P(Ri) = TRUE (predicatul este satisfacut pentru regiunea Ri)

şi P(Rj) = TRUE *atunci*,

daca P(Ri U Rj) = TRUE

=> Ri si Rj sunt regiuni similare si trebuie să fie unificate.

- Algoritmul „split & merge" este o procedură recursivă care include în fiecare iteraţie divizare şi unificare :

- dacă o regiune este neomogenă (P(R) = FALSE), ea este spartă în 4 regiuni.
- altfel

  dacă R este similara cu o regiune adiacenta ($P(R_i$ U R) = TRUE), se unifica R cu R_i

Avantaj:

Algoritmul produce un numar de regiuni mai mic si regiuni mai compacte decât algoritmul bazat numai pe divizare.

Segmentarea prin divizare şi unificare(2)

Procedura *split & merge* :

- **dacă regiunea curentă, R, nu este omogenă**, este divizată în 4 regiuni şi procedura este apelată recursiv pentru fiecare regiune;

altfel (regiunea R este omogenă):

1. Se calculează media regiunii R, mR; unifica = FALSE;
2. Pentru fiecare regiune Ri adiacenta cu R (pe cele 4 laturi ale regiunii R)

   daca |mR – mRi|< prag atunci

   - unifica = TRUE
   - calculeaza media M, a intensitatilor pixelilor din regiunile R si Ri
   - extinde regiunea Ri cu regiunea R si asigneaza regiunii Ri val. de intensitate M
3. Daca unifica == FALSE (nici o unificare nu a fost posibila)

   - se creaza o noua regiune, R si I se asigneaza valoarea de intensitate mR.

In pasul 2: dacă mai multe unificări sunt posibile, se poate alege cea mai bună unificare, adică unificarea cu regiunea vecină care are intensitatea cea mai apropiată de valoarea medie a regiunii curente.

Segmentarea prin divizare şi unificare(3)

Notă privind implementarea:

- Regiunea curenta poate fi inconjurata de:

 - pixeli care apartin unor regiuni adiacente

 - pixeli care nu au fost alocati unei regiuni

- Fiecare regiune creata este etichetata:

  - se foloseste o matrice imagine in care se atribuie fiecarui pixel al regiunii eticheta regiunii

  - se creaza o lista a regiunilor create si pentru fiecare regiune:

    - eticheta regiunii

    - media intensitatilor

  - la unificarea a 2 regiuni:

    - se atribuie pixelilor regiunii obtinute prin unificare aceeasi eticheta: se extinde regiunea adiacenta cu regiunea curenta

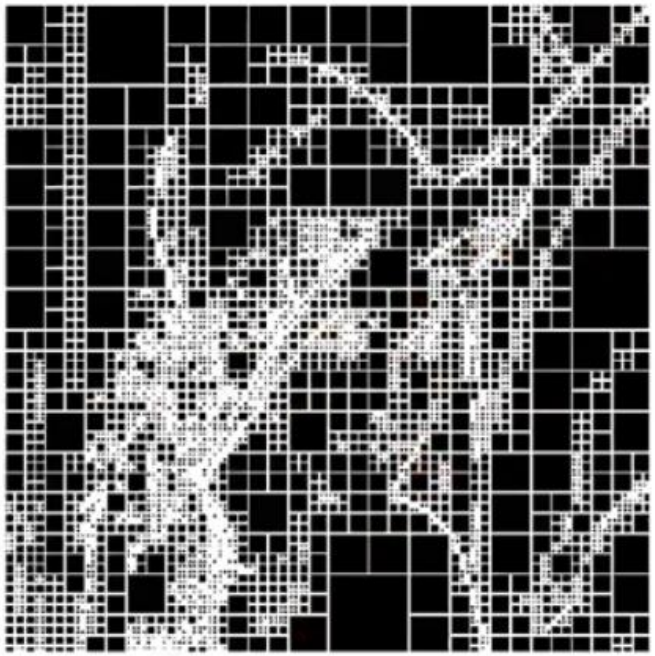
    - se asigneaza noii regiuni media intensitatilor noii regiuni

Segmentarea prin divizare şi unificare(4)

Avantaje:

- Se poate alege nivelul de divizare pe baza criteriului de omogenitate
- Criteriul de unificare poate fi diferit de cel de divizare
- Prin cele 2 criterii se poate controla suprasegmentarea imaginii

Dezavantaj: poate produce regiuni rectangulare



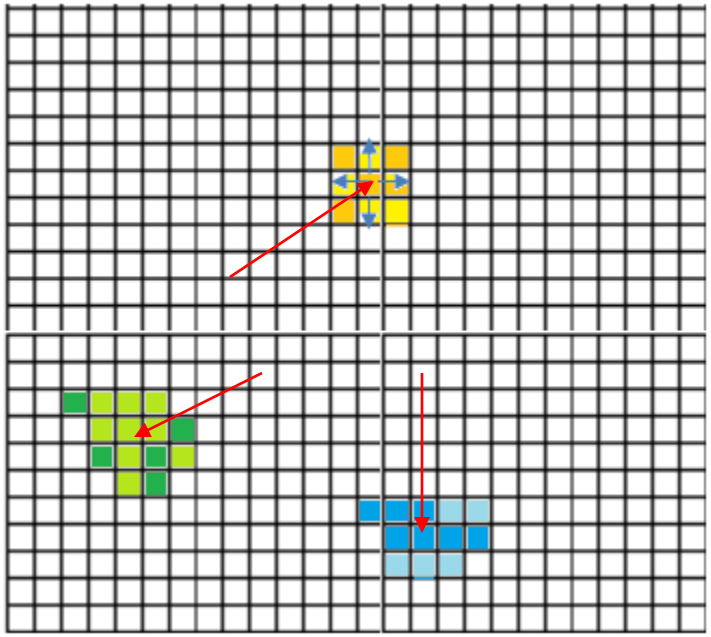
Segmentare prin extindere folosind pixeli germen (1) (Region growing)

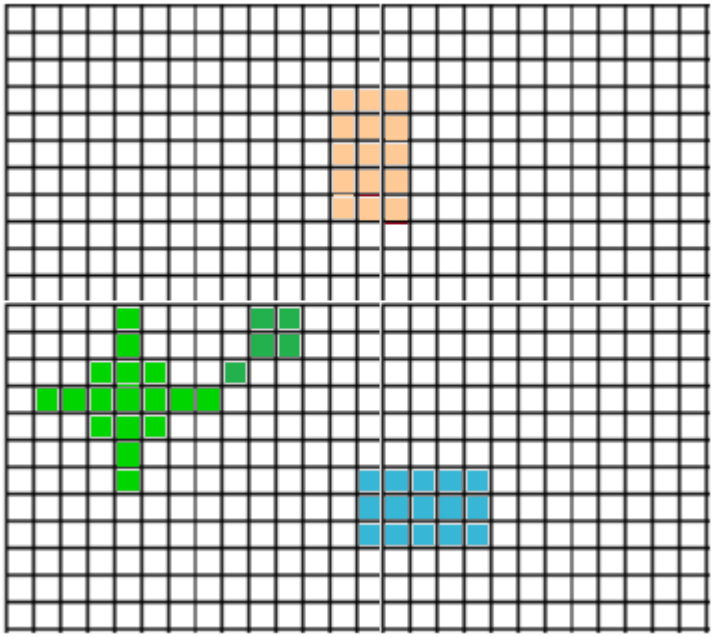
❖ Se pleacă de la pixeli „germen” în jurul cărora se vor forma regiunile.

❖ Plecând de la pixelii germen se formează în jurul lor regiuni adăugând iterativ pixelii vecini care satisfac criteriul de similaritate.

❖ Segmentarea prin extindere se bazează pe :

- O regulă care descrie mecanismul de extindere
- Un criteriu de similaritate, care restrictioneaza extinderea regiunilor





Segmentare prin extindere folosind pixeli germen (2)

Mecanismul de extindere

repeta

clasificat = false

pentru fiecare regiune existenta in pasul k, $R_i^{(k)}$ cu i = 1, ... N

pentru fiecare pixel X, neclasificat, din vecinătatea de 8 (sau 4) pixeli a fiecărui pixel de pe frontiera regiunii

daca P ($R_i^{(k)}$ U $\{X\}$) = TRUE, unde P este criteriul de similaritate, **atunci**:

- clasificat = true
- clasifica pixelul X adaugandu-l la regiunea $R_i^{(k)}$
- recalculeaza media regiunii $R_i^{(k)}$

pana (!clasificat)

In imaginea segmentata, toti pixelii unei regiuni au ca valoare intensitatea medie a regiunii.

Segmentare prin extindere folosind pixeli germen (3)

Criteriul de similaritate, P, poate fi:

$P(R_i^{(k)} \cup \{X\}) = |$intensitate pixel X – intensitatea medie a regiunii $R_i^{(k)}| < T$

T este un prag global.

- Algoritmul se termină atunci când într-o iteraţie numărul de pixeli clasificaţi nu s-a modificat (nici o regiune nu s-a mai putut extinde).

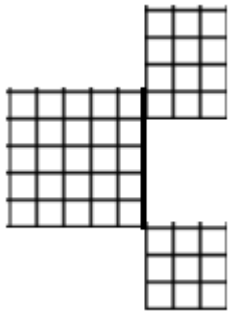
- Pixelii germen pot fi aleşi interactiv de către utilizator, sau automat ca pixeli a căror intensitate/nuanță corespunde vârfurilor histogramei.

- Performanţa algoritmului depinde mult de alegerea pixelilor germen.
- Rezultatele algoritmului pot fi diferite pentru pixeli germen diferiti.

Unificarea regiunilor (merging)

- In urma segmentării prin extindere („creşterea” regiunilor) pot rezulta **regiuni adiacente cu proprietăţi statistice similare.** Ele pot fi reunite printr-un algoritm de unificare („merging”).
- Ca proprietăţi statistice se pot folosi media şi deviaţia standard.

Pentru o regiune Ri de n pixeli, dintr-o imagine f(i,j):



media: $$m_i = \frac{1}{n} \sum_{(k,l) \in R_i} f(k,l)$$

deviatia standard: $$\sigma_i = \sqrt{\frac{1}{n} \sum_{(k,l) \in R_i} (f(k,l) - m_i)^2}$$

- **Criteriul de unificare** a doua regiuni adiacente, R1 şi R2:

mediile lor sunt suficient de apropriate si deviatiile standard sunt mici:

$$|m_1 - m_2| < K * \sigma_i \qquad \text{cu i = 1,2}$$

- Unificarea poate fi înglobată în mecanismul de extindere descris anterior:

  - dupa fiecare pas de extindere se incearca unificarea regiunii extinse cu regiunile vecine.

Segmentare prin extindere si unificare folosind pixeli germen

repeta

clasificat = false

pentru fiecare regiune existenta in pasul k, $R_i^{(k)}$ cu i = 1, ... N

pentru fiecare pixel X, neclasificat, din vecinătatea de 8 (sau 4) pixeli a fiecărui pixel de pe frontiera regiunii

daca P ($R_i^{(k)}$ U $\{X\}$) = TRUE, unde P este criteriul de similaritate, **atunci**:

- clasificat = true
- clasifica pixelul X adaugandu-l la regiunea $R_i^{(k)}$
- recalculeaza media regiunii $R_i^{(k)}$
- **pentru** fiecare regiune Ra adiacenta cu regiunea $R_i^{(k)}$

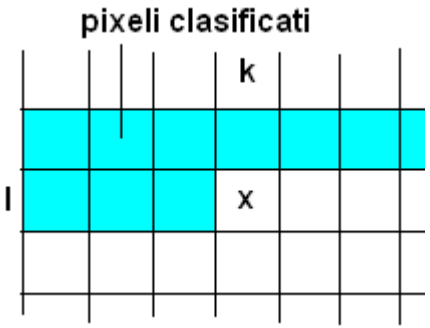
daca Unif (Ra, $R_i^{(k)}$)=TRUE, unde **Unif este criteriul de unificare**, **atunci**

- clasifica pixelii regiunii adiacente Ra adaugandu-i la regiunea $R_i^{(k)}$
- elimina regiunea Ra
- recalculeaza media regiunii $R_i^{(k)}$

pana (!clasificat)

Segmentare prin extindere si unificare (1)
(Region growing and merging)

- Nu sunt disponibile informaţii despre imagine (nu se dau pixeli germen).
- Imaginea este parcursă pe linii şi fiecare pixel este atribuit unei regiuni (clasificat).



- Atunci când s-a ajuns în pixelul (k,l), toţi pixelii de pe liniile anterioare au fost deja clasificati. Deci pixelul (k,l) are regiuni adiacente dacă nu este primul (k = 0, l = 0).
- Se încearcă atribuirea pixelului (k, l) la una dintre regiunile vecine (stanga, sus, stanga-sus).
- Dacă nu poate fi atribuit unei regiuni existente, este atribuit unei regiuni noi (se creează o nouă regiune).
- Performanţa algoritmului este dependentă de <u>regula de extindere</u>, adică de modul de atribuire a pixelului curent la o regiune existentă : P(Ri U {(k,l)}) = TRUE

Segmentare prin extindere si unificare (2)

Regula de asignare se poate baza pe media aritmetică şi deviaţia standard - asignarea este permisă dacă intensitatea pixelului, f(k,l), este apropiată de media Mi a unei regiuni vecine, Ri

P(Ri U {(k,l)}) = $\left| f(k,l) - M_i \right| \le T_i(k,l)$ unde $T_i(k,l) = (1 - \frac{\sigma_i^{'}}{M_i^{'}}) * T$

- M'_i şi σ'_i sunt media şi deviaţia standard ale regiunii Ri la care s-a adaugat pixelul (k,l)
- T este pragul global
- Daca P(Ri U {(k,l)}) = TRUE pentru mai multe regiuni vecine Ri, atunci pixelul este atribuit regiunii Ri pentru care diferenţa $\left| f(k,l) - M_i \right|$ este minima

Creşterea regiunilor este dependentă de parametrul T:

- dacă T este mic, pragul Ti(k,l) este mic pentru toate regiunile şi extinderea regiunilor este dificilă → vor rezulta multe regiuni mici.

- dacă T este mare, pot rezulta regiuni puţin omogene şi cu deviaţii standard mari.

Segmentare prin extindere si unificare (3)

Pragul $T_i(k,l) = (1 - \frac{\sigma_i^{'}}{M_i^{'}}) * T$ depinde, de asemenea, de raportul $\frac{\sigma_i^{'}}{M_i^{'}}$

- **Pentru regiuni omogene**: $\frac{\sigma_i^{'}}{M_i^{'}}$ tinde catre zero si Ti tinde catre T

  Regula de asignare a pixelului (k,l) la o regiune: $|f(k,l) - M_i| \leq T_i(k,l)$

  - T desemneaza diferenţa maxima permisa între intensitatea unui pixel si media unei regiuni foarte omogene, la care poate fi asignat pixelul.

- **Pentru regiuni mai putin omogene**: $\frac{\sigma_i^{'}}{M_i^{'}}$ este mai mare si Ti devine mai mic decat T

  - Extinderea unor astfel de regiuni este restrictionata.

Segmentarea bazată pe praguri (1)

Se modifică intensitatea fiecărui pixel din imaginea de intrare în funcție de intervalul de intensități în care se află.

Exemplu: segmentarea în N intervale de intensitate egale

f(x, y) – imaginea de intrare şi g(x,y) – imaginea de ieşire (segmentată)

N – numarul de intervale de intensitate

L - nr. de nivele de intensitate din imagine

$$g(x, y) = \begin{cases} T_i, & \text{dacă } T_i <= f(x,y) < T_{i+1} \\ T_{N-1}, & \text{dacă } T_{N-1} <= f(x,y) < L \end{cases} \qquad T_i = i*L/N, \ i=0,1,.., N-2$$

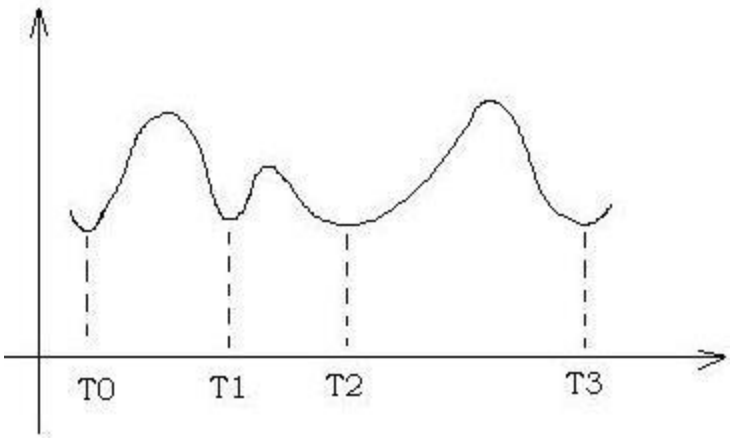
$T_i = i*L/N$, $i = 0, 1, 2,\ldots N-1$ se numesc praguri

Exemplu: N = 4, L = 256, L/N = 64 → pragurile: 0, 64, 128, 192

$$g(x, y) = \begin{cases} 0, & \text{daca } 0<= f(x,y) < 64 \\ 64, & \text{daca } 64<= f(x,y) < 128 \\ 128, & \text{daca } 128<= f(x,y) < 192 \\ 192, & \text{daca } 192<= f(x,y) < 256 \end{cases}$$

Segmentarea bazată pe praguri (2)

Valorile de prag pot fi alese ca puncte de minim local ale funcţiei histogramă :



Pragurile T_i pot fi alese interactiv sau automat

- Dacă f(x, y) este funcţia imagine de intrare, atunci imaginea segmentată este definită de :

  $g(x, y) = R_i$ dacă $T_{i-1} <= f(x,y) < T_i$ cu i = 1,N

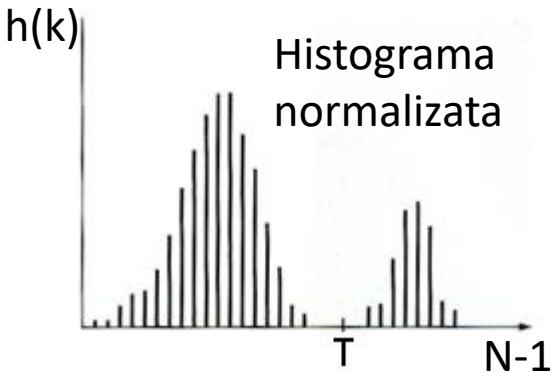
  R_i poate fi intensitatea medie a pixelilor care satisfac conditia.

- Metoda poate fi aplicata si pentru imagini color, construind histograma pt componenta Hue.

- Functia histograma nu este o functie derivabila: este o functie definita pt valori discrete

  → se folosesc diverse metode pentru a aproxima minimele functiei histogramă

Calculul unui prag global pe baza histogramei



Segmentarea pe baza unui prag global:

g(x,y) = 1 daca f(x,y) >T

0 daca f(x,y) <=T

Calculul iterativ al pragului T

1. Se alege un prag T estimat, de regula valoarea mediană a nivelurilor de gri din imagine

   T împarte pixelii imaginii in 2 grupuri:

   G1, cu nivele de gri <=T; G2, cu nivele de gri >T

2. Se calculeaza media nivelelor de gri din G1, μ1, si cea a nivelelor de gri din G2, μ2

$$\mu 1 = \sum_{k=0}^{T}(k * h(\mathrm{k}))/\sum_{k=0}^{T} h(\mathrm{k}) \qquad \mu 2 = \sum_{T+1}^{N-1}(k * h(\mathrm{k}))/\sum_{T+1}^{N-1} h(\mathrm{k})$$

3. Se calculeaza un nou prag:

$$T = (\mu 1 + \mu 2)/2$$

4. Se repetă paşii 2-3 până când diferenta dintre pragurile T calculate în 2 iteratii succesive este mai mică decât o limită predefinită.

Alegerea pragului optim pe baza histogramei (1)

- Calculul lui T pe baza algoritmului anterior tinde sa produca 2 grupuri cu un numar aproximativ egal de pixeli.

Fie μ1(T) şi σ1(T) media şi variatia intensitatilor din grupul G1 (nivele <=T)

Fie μ2(T) şi σ2(T) media şi variatia intensitatilor din grupul G2 (nivele >T)

Omogenitatea unui grup se poate măsura prin ***variația în interiorul grupului***.

- **Pragul optim este acela care minimizează *variația în interiorul fiecărui grup*, maximizând astfel omogenitatea fiecărui grup.**
- Fie p1(T) şi p2(T) probabilitatile celor 2 grupuri:

  p1(T) = (1/n) $\sum_{k=0}^{T} h$(k) şi p2(T) = (1/n) $\sum_{k=T+1}^{N-1} h$(k), n este numarul de pixeli din imagine
- **Variatia în interiorul fiecarui grup** este:

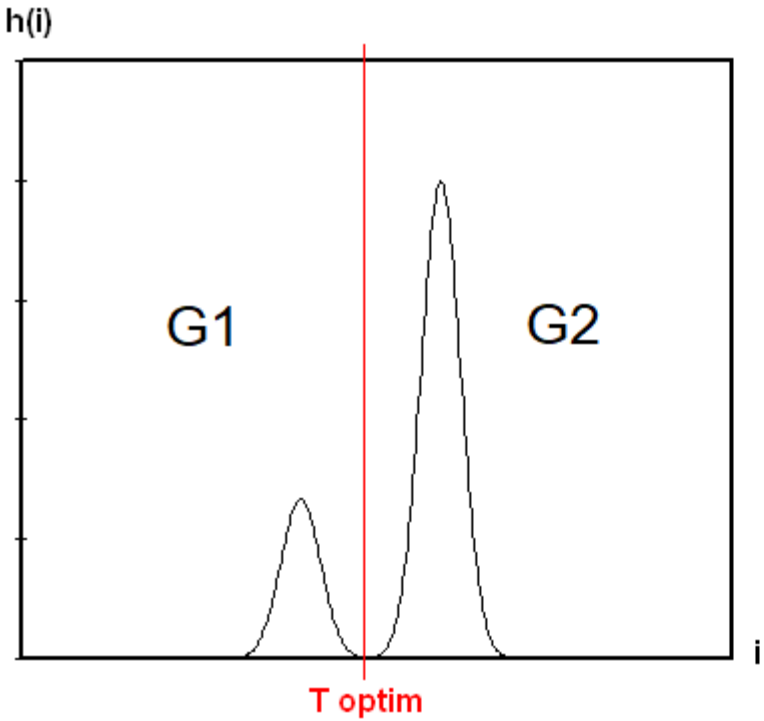
  $\sigma 1^2$ (T) = (1/(n * p1(T)) $\sum_{k=0}^{T}[\mathrm{h(k)} * (\mathrm{k} - \mu 1(\mathrm{T})]^2$

  $\sigma 2^2$ (T) = (1/(n * p2(T)) $\sum_{k=T+1}^{N-1}[\mathrm{h(k)} * (\mathrm{k} - \mu 2(\mathrm{T})]^2$

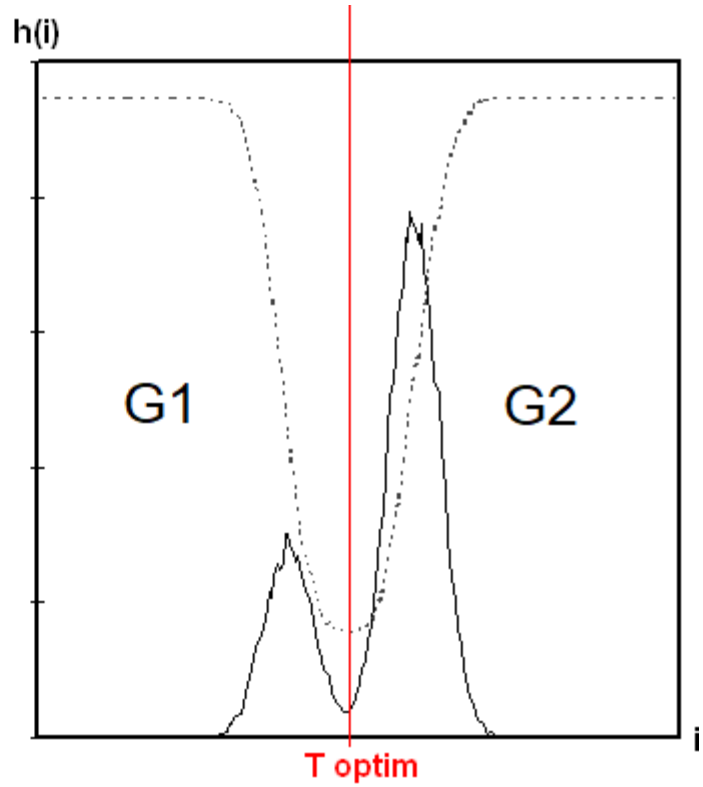
Alegerea pragului optim pe baza histogramei (2)

$\sigma w^2 (T) = \sigma 1^2 (T)*p1(T) + \sigma 2^2 (T)*p2(T)$

- **Pragul optim T este nivelul de intensitate pentru care expresia σ_w^2 are valoarea minima**
  - **L evaluari ale expresiei σ_w^2 pentru o imagine cu L nivele de gri.**



Histograma idealizata a unei imagini obiect/fond.

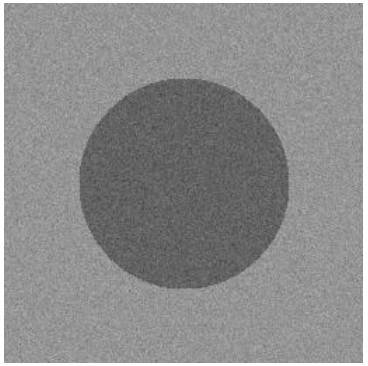


——— Histograma

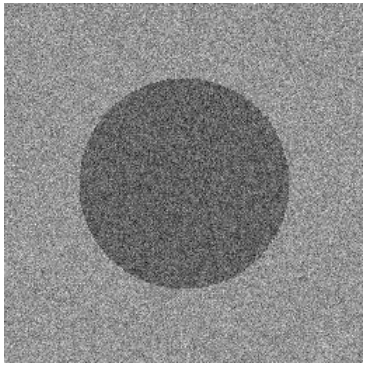
- - - - - - - Omogenitatea

Alegerea pragului optim pe baza histogramei (3)

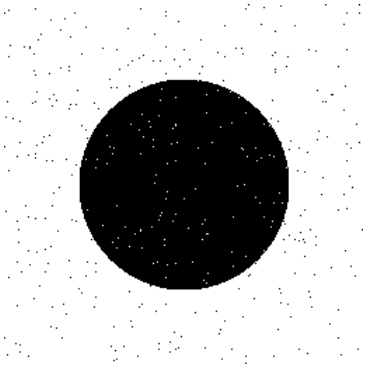
Rezultatul segmentarii este influentat de prezenta zgomotului în imagine:



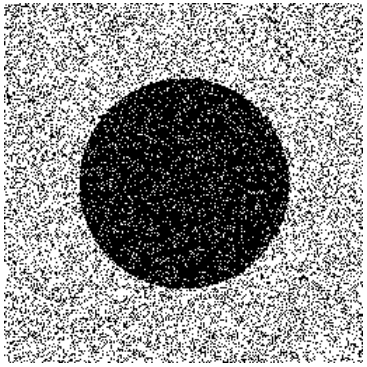
Imagine cu zgomot redus



Imagine cu nivel de zgomot înalt



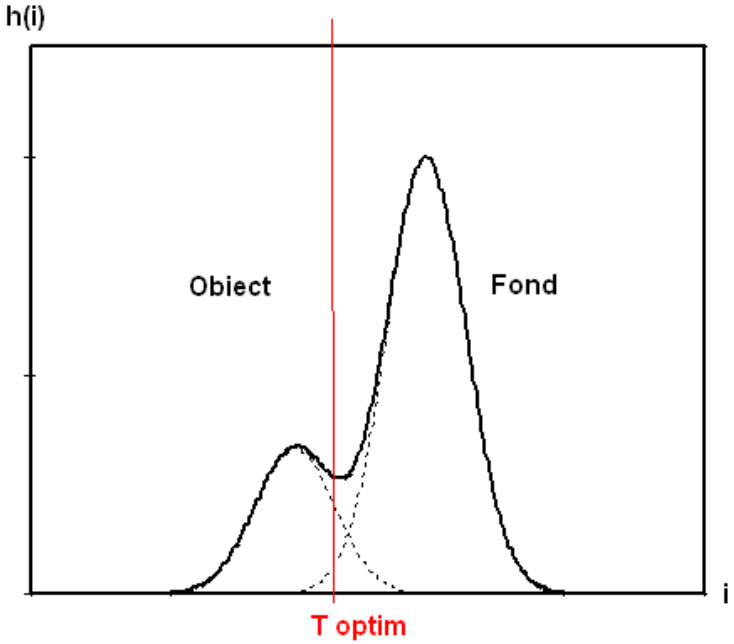
Rezultatul segmentarii cu pragul (T = L/2)



Rezultatul segmentarii cu pragul (T = L/2)

Alegerea pragului optim pe baza histogramei (4)

Cu cât suprapunerea nivelurilor de gri ale celor doua grupuri este mai mare cu atat clasificarea pixelilor este mai eronata.



Metoda se poate generaliza pentru alegerea mai multor praguri optime. Exemplu:

- **pentru 2 praguri**, 3 grupuri: T1, T2; G1 (0<=nivel<T1), G2 (T1<=nivel<T2), G3(T2<=nivel<L-1)

Expresia de minimizat devine:

$$\sigma w^2 (T1,T2) = \sigma 1^2 (T1)*p1(T1) + \sigma 2^2 (T1,T2)*p2(T1,T2) + \sigma 3^2 (T2)*p3(T2)$$

- Gasirea pragurilor optime T1, T2: complexitate mare
- O metoda de segmentare mai eficienta pt mai multe praguri: segmentarea prin cresterea regiunilor

Segmentarea bazată pe praguri- concluzii (1)

- Segmentarea prin aplicarea de praguri uniforme (i*L/N):

- Nu dă rezultate bune dacă histograma imaginii este concentrată într-un domeniu mic de intensităţi, deoarece majoritatea pixelilor vor fi repartizaţi într-un număr foarte mic de regiuni

- În astfel de cazuri, segmentarea prin aplicarea de praguri uniforme se poate face dupa ce asupra imaginii de intrare se aplică transformarea de egalizare a histogramei.

- Alegerea automată a pragurilor pe baza histogramei:

- În multe cazuri, „lobii” histogramei nu sunt clari datorită contribuţiei pixelilor aflaţi la frontiera dintre două regiuni. Dacă se elimină sau se moderează contribuţia acestor pixeli în calculul histogramei, atunci lobii vor fi mai clari.

  – Metoda cea mai simplă de modificare a histogramei în acest scop este de a detecta pixelii de front din imaginea de intrare, excluzându-i apoi în calculul histogramei.

- Calculele necesare pentru alegerea automata a mai multor praguri optime sunt complexe

Segmentarea bazată pe praguri- concluzii (2)

Dezavantajele segmentarii bazate pe praguri:

- Segmentarea bazată pe aplicarea de praguri ține cont de omogenitatea regiunilor obtinute, nu şi de conectivitatea lor, deoarece aplicarea pragurilor se face pixel cu pixel, fără să se ţină cont de vecinătatea fiecărui pixel. Pot rezulta mai multe regiuni cu acelasi nivel de intensitate neconectate (ar putea fi parti ale aceluiasi obiect).
- Rezultatul segmentarii bazate pe praguri poate fi supus unei analize de conectivitate a regiunilor pentru detectia regiunilor conectate.

Segmentarea imaginilor color (1)

- Oricare dintre algoritmii de segmentare pt imagini in niveluri de gri poate fi folosit pt segmentarea imaginilor color.
- In functie de caracteristicile aplicatiei se pot obtine rezultate mai bune segmentand intr-un anumit spatiu de culori:
  - HSV, HSI: permit segmentarea pe baza nuantei (H), a saturatiei (S) sau a intensitatii (V/I)
    - Se pot folosi agoritmi bazati pe regiuni sau algoritmi de clasificare la nivel de pixel (thresholding, clustering, s.a)
    - Segmentarea dupa H nu da rezultate bune daca S are valori mici

      → segmentare folosind H+S
  - RGB este sensibil la variatii ale luminozitatii: rezultatul segmentarii depinde de luminozitatea la momentul achizitiei imaginii.

Segmentarea imaginilor color (2)

- **YCbCr** este un spatiu de culori mai putin sensibil la variatii ale luminozitatii:
  - Y reprezinta luminanta (stralucirea) iar Cb si Cr cromaticitatea ca functii de Y

  Y = 0.2126 *red+0.7152*green+0.0722*blue;

  Cb = 0.5389*(blue-Y); Cr = 0.6350*(red-Y)

  - Modelul se foloseste in compresia imaginilor video (JPEG, MPEG):
    - Ochiul uman este mai sensibil la luminozitate (Y) → se reprezinta cu o precizie mai mare
  - Segmentarea se poate face folosind Y sau cromaticitatea
  - Se foloseste pentru segmentarea pielii, segmentarea imaginilor satelitare, s.a.

Segmentarea imaginilor de adancime (1)

- In prezent exista multe camere care produc imagini de adancime:
  RGB-D: Kinect 1, Kinect 2
  D: Structure sensor, Intel RealSense, s.a.
- Camerele stereo pot produce imagini de adancime
- Adancimea: importanta in reconstructia 3D a spatiului din imagine

O imagine RGB-D are 4 canale: R, G, B şi D (depth) adancime.

- Intr-o imagine de adâncime, D(x,y) reprezinta distanta (D) de la cameră la punctul din mediul real proiectat in pixelul (x,y).
- D este măsurată în spațiul camerei (In OpenGL sistemul de coordonate observator).
- D este reprezentata printr-un nivel de gri, crescator cu distanta, zonele fara informatie de adancime fiind reprezentate prin pixeli albi.
- Lipsa informatiei de adancime:
  - Orice camera RGB-D “vede” intr-un anumit interval de adancime: Dmin- Dmax
  - Poate avea si alte cauze: luminozitatea, defecte ale camerei, etc.

Segmentarea imaginilor de adancime (2)

Segmentarea unei imagini D – numai pe baza adancimii fiecarui pixel

Segmentarea unei imagini RGB-D – pe baza adancimii/ a culorii/ a adancimii si culorii

Tehnologii prin care se obtine informatia de adancime:

- Structured light: Kinect 1, Structure sensor – dispozitivul proiecteaza pe scena 3D un şablon (de regula o grila de bare orizontale); adancimea fiecarui punct din scena 3D se deduce pe baza deformării şablonului.
- TOF (Time Of Flight): Kinect 2 – adâncimea în scena 3D a unui punct se obţine prin măsurarea intervalului de timp în care un semnal luminos parcurge distanţa de la cameră până la punct şi înapoi.

O prezentare tehnica detaliata a tehnologiilor “Structured light” si “TOF” poate fi consultata la: https://arxiv.org/pdf/1505.05459.pdf

Imagini de adancime

- Probleme cu informatia de adâncime

  - zgomot: eroarea creste odata cu adancimea

  - lipsa informatiei (in medii puternic luminate de soare, pe suprafete reflectante)

Componenta RGB a unui cadru imagine RGB-D

Componenta D (harta de adâncime) a cadrului: adâncimea reprezentata prin niveluri de gri

Zonele albe sunt zone cu informatie lipsa

Nor de puncte

- Harta de adâncime poate fi transformată într-un ”nor de puncte 3D”, (x,y,z), raportat la spatiul 3D al camerei, în care:
  - Coordonata (0,0,0) corespunde pozitiei camerei
  - Fie P(xi, yi) punctul din norul de puncte corespunzator unui pixel (xi,yi).
  - Coordonatele (x,y,z) ale punctului P(xi,yi) se obtin folosind informatia de adancime a pixelului (xi,yi) – valoarea pixelului - si proprietatile intrinseci ale camerei (distanța focală, punct central, s.a).
- Exista o corespondenta 1 la 1 intre pixelii din imagine si punctele 3D din norul de puncte.
- Daca doi pixeli sunt vecini, atunci se poate considera ca punctele lor 3D corespondente sunt vecine daca adancimea (valorile) pixelilor nu difera foarte mult.

Segmentarea planara

- Segmentare in regiuni plane.
- Adecvata pentru imagini de interior, in care multe suprafete sunt plane: podeaua, peretii, suprafete de mobilier.
- Punctele unei suprafete plane au aceeasi normală → segmentarea se poate face:
  - pe baza normalelor locale, calculate în punctele norului de puncte
  - ținând cont de adancimea punctelor (suprafetele plane paralele situate la adâncimi diferite au normalele identice)
- Imaginile de adancime au mult zgomot si informatie lipsa, de aceea normalele nu pot fi calculate exact → diferite aproximari pentru a compensa aceste defecte.
- Multe aplicatii necesita extragerea informatiei din imagini in timp real → se fac compromisuri intre precizie si timp de calcul.

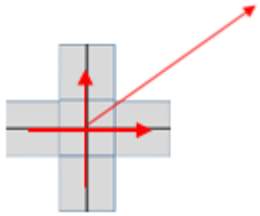
Calculul normalelor (1)

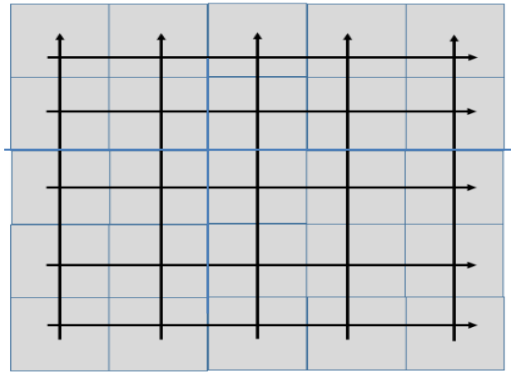
Algoritm rapid de calcul al normalelor locale (in punctele norului de puncte)

- Fie (xi,yi) un pixel al imaginii de adancime şi P(xi,yi) punctul corespunzator in spatiul 3D
- Se calculeaza 2 vectori perpendiculari, tangenti la suprafață in punctul P (aproximare prin diferente finite):

  - un vector care uneste punctele P(xi-1,yi) şi P(xi+1,yi)
  - un vector care uneste punctele P(xi,yi-1) şi P(xi,yi+1)
- Se calculeaza normala in P(xi,yi) ca produs vectorial al celor 2 vectori
- Pentru a compensa zgomotul si informatia lipsa, cei 2 vectori tangenti se pot aproxima prin media vectorilor tangenti calculati pe vecinatățile lui P(xi,yi) de 3x 3 pixeli şi 5x5 pixeli.

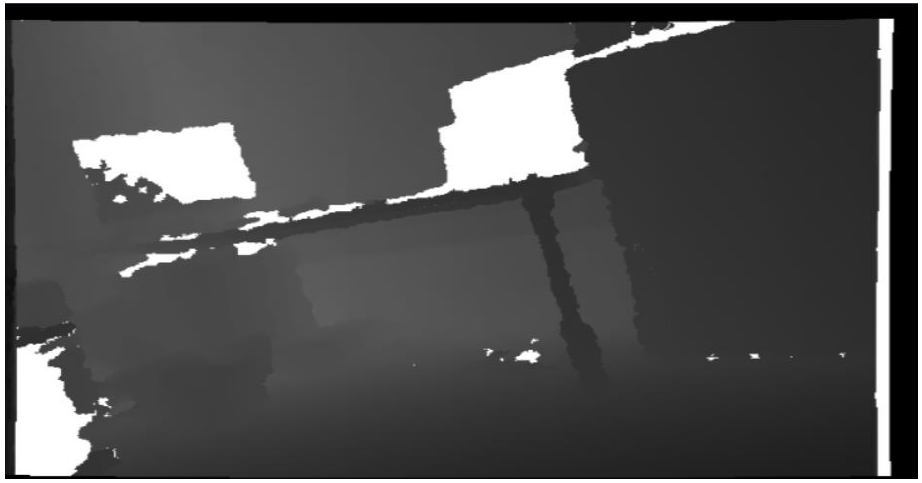
N(P(xi,yi))



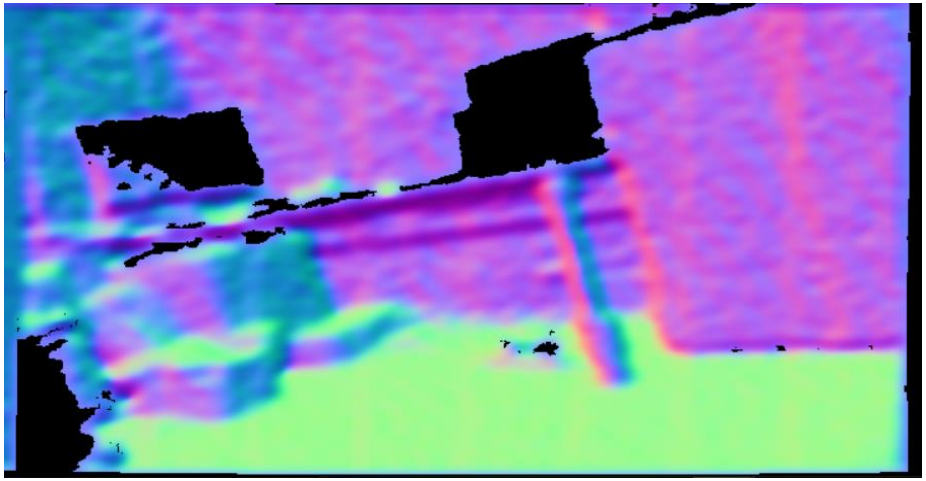


Calculul normalelor (2)

- Normalele normalizate au componentele Nx,Ny,Nz cuprinse intre -1 si 1.
- Pentru vizualizarea hartii de normale, componentele Nx,Ny,Nz se transforma in componente R,G,B cu valori in intervalul [0,1]: (R,G,B) = ((Nx,Ny,Nz) + (1,1,1))/2;



Harta de adancime

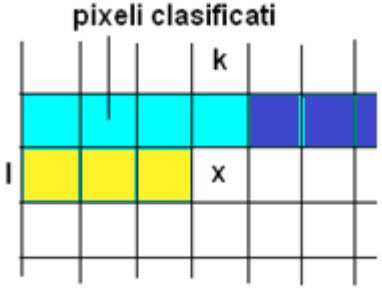


Harta de normale: zona verzuie indica o suprafata orizontala (axa OY – in sus)

N =[0,1,0] → (0.5, 1, 0.5)

Zonele negre – fara informatie (nu s-a putut calcula normala)

Algoritm de segmentare planara



Segmentare prin extindere si unificare, bazata pe parcurgerea imaginii de adancime linie cu linie.

Pasul 1:

- **Crearea unui set de regiuni prin extindere si unificare**

  - o regiune planara: un set de pixeli conectat cu adancimi si normale similare

  - **criteriul de extindere**: pixelul adaugat are normala apropiata de normala medie a regiunii extinse si adancimea apropiata de cea a pixelului vecin care face parte din regiunea extinsa

- **Pixelii unei regiuni rezultate din pasul 1: au aceeasi normala si aceeasi adancime** (media normalelor/adancimilor pixelilor regiunii) → o regiune corespunde unei suprafete 3D

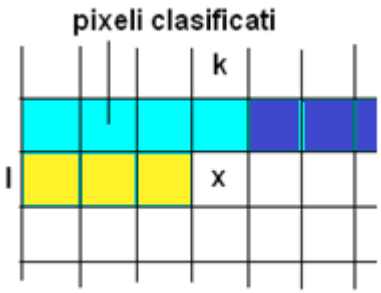
- **Crearea informatiei de adiacenta a regiunilor** (pentru fiecare regiune se memoreaza identificatorii regiunilor adiacente)

- Numarul de regiuni rezultate depinde de pragurile folosite în extindere

Pasul 2: **Se unifica regiunile adiacente similare create in primul pas**

  – Rezulta regiuni planare mari

Algoritm de segmentare planara - pasul 1



- Pixelul curent se compara cu
  - Pixelul vecin de sus, din stanga sus, din stanga
- Pentru fiecare vecin, se calculeaza un cost al unificarii

pixelului curent cu regiunea din care face parte vecinul, in functie de:

  - Diferenta între adancimea pixelului curent si adancimea pixelul vecin
  - Diferenta între normala punctului 3D corespunzator pixelului curent si normala medie a regiunii din care face parte pixelul vecin
  - Eroarea datorata adancimii pixelului (creste cu adancimea)
- Pixelul curent se ataseaza la regiunea din care face parte pixelul vecin cu cost minim, daca acest cost este mai mic decat un prag dat. Se calculeaza normala medie a regiunii extinse cu pixelul curent.
- Daca pixelul curent nu poate fi atasat la niciuna dintre regiunile vecine, se creaza o regiune noua.