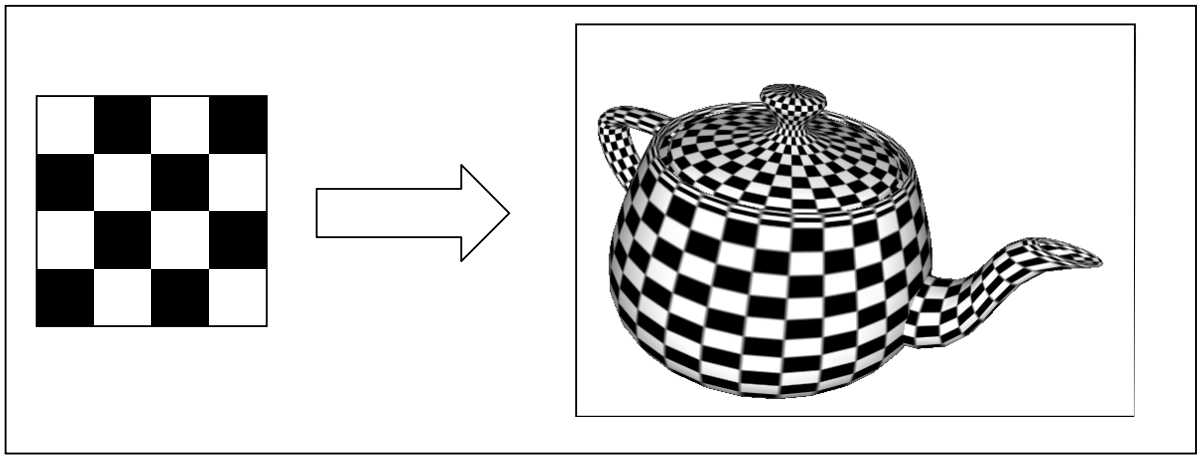
Textură

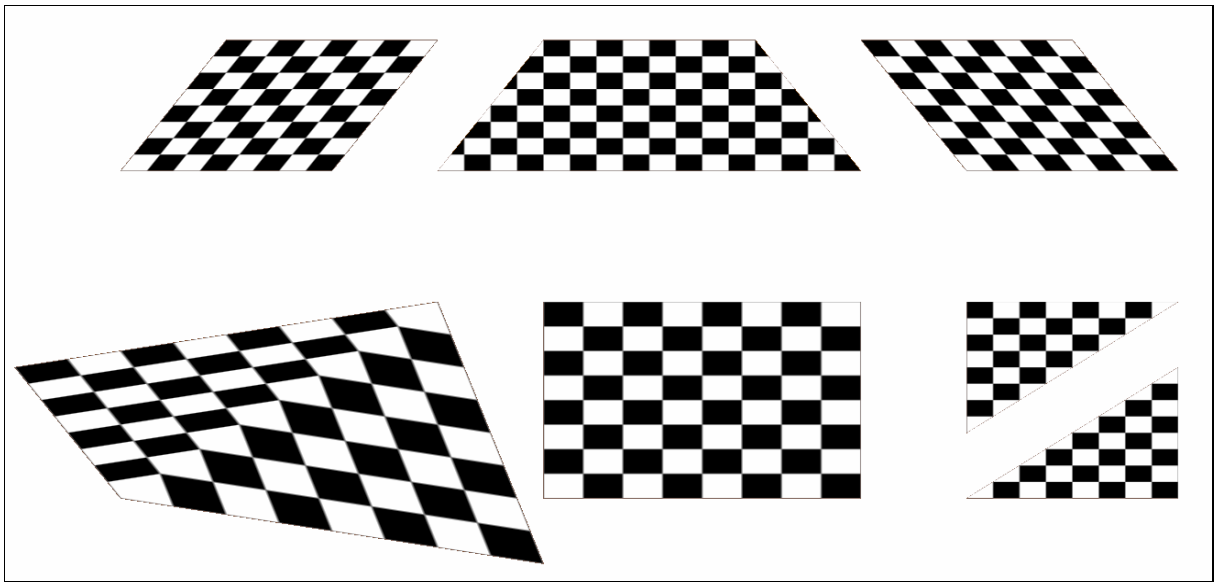
Textura este o imagine care se aplică peste un obiect grafic. Imaginea folosită la această operație poate să fie:

- **generată** prin calcul
- **citită dintr-un fişier** (există diverse formate de memorare a imaginii: bmp, jpg, gif, png, etc.)

Un exemplu de imagine aplicată peste un obiect se vede în figura următoare.



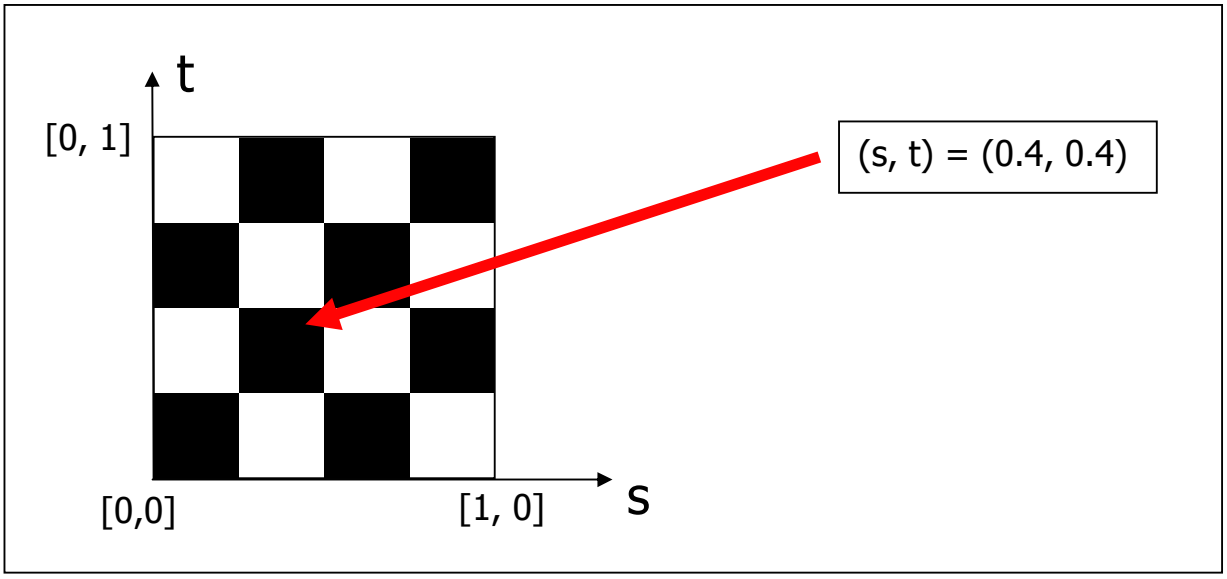
In figura de mai jos se văd alte exempleu de folosire a texturii. Pentru prima imagine din rândul al doilea se observă un mod eronat de aplicare a texturii (imaginea apare deformată).



O imagine (citită sau calculată) trebuie "**pregătită" ca o textură pentru OpenGL** (trebuie memorată în memoria internă, sub o anumită formă).

Texturile pot fi considerate ca **matrici de pixeli**, sau **matrici de texeli** (**textel** = element din textură)**.** In cele mai multe cazuri dimensiunile unei texturi (numărul de linii şi numărul de coloane) sunt puteri ale lui 2 (această restricție este cerută de OpenGL). Putem avea

astfel texturi cu dimensiunile 64x64 pixeli, 128x128, 256x128, etc. Dacă se memorează aceste matrici, atunci este necesar un spațiu mare de memorie internă. Indiferent de dimensiunile inițiale ale imaginii, pentru referire în comenzile OpenGL, pe fiecare dimensiune se face o **normalizare** a texturii şi va rezulta o imagine cu coordonate în patratul **[0,1]*[0,1]**. In acest caz referirea la un pixel din textură se poate face prin coordonatele **(s,t),** fiecare din coordonate este în intervalul [0,1].



Important. La calcularea sau citirea imaginilor trebuie avută în vedere observația că **într-o imagine** coordonatele (0,0) sunt în stânga-sus, iar **într-o textură** aceste coordonate sunt în stânga-jos.

Generarea unui şir de octeți (pentru textură) prin calcul:

```
    byte T[] = new byte[n*n*3];  //textura, o culoare se da pe 3 octeti
    int c;
    int k=16;//32;
    //se construieste vectorul T, cu culorile pixelilor, linie dupa linie
    //pentru fiecare pixel se folosesc 3 octeti
    for (i=0;i<n;i++){
      for (j=0;j<n;j++){
        c = (int)(i/k)+(int)(j/k);
        if (c==2) c=0;
        c=c*255;   //=>c=0 sau c=255
        T[(j*n+i)*3]   = (byte)c;  //red
        T[(j*n+i)*3+1] = (byte)c;  //green
        T[(j*n+i)*3+2] = (byte)c;  //blue
      }
    }
```

Generarea unui şir de octeți (pentru textură) prin citirea unei imagini:

```
Color c;
//se parcurge toata imaginea "img" citita anterior
for (i=0;i<img.getWidth();i++){
  for (j=0;j<img.getHeight();j++){
    c = new Color(img.getRGB(i,j));
    //k=(j*m+i)*3; //cu ac.atribuire va rezulta textura "inversata"
    k=((n-1-j)*m+i)*3;
    //imaginea initiala are (0,0) in stanga-sus
```

```
    //prin scrierea pixelilor in aceasta ordine se face o inversare a imaginii
    //(ca sa fie aplicata mai usor pe obiecte, in OpenGL: (0,0) e la stanga-jos)
     T[k] = (byte)c.getRed();
     T[k+1] = (byte)c.getGreen();
     T[k+2] = (byte)c.getBlue();
   }
}
```

Deoarece texturile sunt mari consumatoare de resurse (mai ales de memorie internă), această facilitate (de utilizare a texturilor) poate fi **activată**/**dezactivată**, prin:

```
glEnable(GL_TEXTURE_2D);
glDisable(GL_TEXTURE_2D);
```

Informațiile despre o textură se păstrează într-o zonă de memorie internă, cu o **anumită structură**. Crearea uneia sau a mai multor zone pentru texturi se face prin comanda:

```
glGenTextures(n, t, deplasare)
```

unde **n** este numărul de zone generate, **t** este un vector unde se păstrează adresele zonelor pentru texturile generate, iar **deplasare** este indicele din vector de unde începe memorarea adreselor pentru zonele generate. Exemplu pentru generarea unei singure texturi:

```
glGenTextures(1, IDTextura, 0);
```

iar textura astfel generată se va identifica prin `IDTextura[0]`
Una sau mai multe zone generate anterior pentru texturi se pot elimina cu comanda:

```
glDeleteTextures(n, t, deplasare)
```

Dacă sunt create mai multe zone pentru textură, atunci una dintre zone (una dintre texturi) poate fi **activată** (şi devine **curentă** la operațiile de desenare):

```
glBindTexture(GL_TEXTURE_2D, t)     //t identifică textura curentă
```

Exemplu:

```
glBindTexture(GL_TEXTURE_2D, IDTextura[0]);
```

La un moment dat o singură textură este activată (o singură textură este curentă). Această textură se va folosi la comenzile care urmează şi care gestionează date despre o textură.

Texelii (pixelii din textură, generați prin calcul sau citiți dintr-o imagine), se pot folosi **pentru crearea texturii** prin comanda:

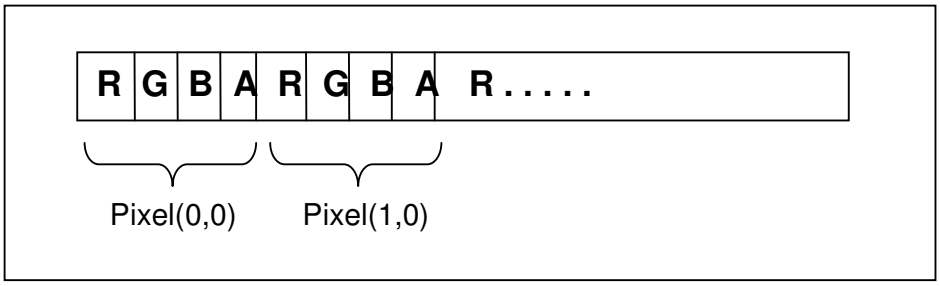
```
glTexImage2D(GL_TEXTURE_2D, nivel, format_intern, latime,
    inaltime, margine, format, tip, pixeli);
```

unde:

- **nivel** - nivelul de rezoluție (in continuare se va folosi nivelul 0)
- **format_intern** - precizează numărul de componente de culoare din textură: **1, 2, 3, 4**, sau una din constantele: `GL_ALPHA`, `GL_ALPHA4`, `GL_ALPHA8`, `GL_ALPHA12`, `GL_ALPHA16`,

GL_COMPRESSED_ALPHA, GL_COMPRESSED_LUMINANCE, GL_COMPRESSED_LUMINANCE_ALPHA, GL_COMPRESSED_INTENSITY, GL_COMPRESSED_RGB, GL_COMPRESSED_RGBA, GL_DEPTH_COMPONENT, GL_DEPTH_COMPONENT16, GL_DEPTH_COMPONENT24, GL_DEPTH_COMPONENT32, GL_LUMINANCE, GL_LUMINANCE4, GL_LUMINANCE8, GL_LUMINANCE12, GL_LUMINANCE16, GL_LUMINANCE_ALPHA, GL_LUMINANCE4_ALPHA4, GL_LUMINANCE6_ALPHA2, GL_LUMINANCE8_ALPHA8, GL_LUMINANCE12_ALPHA4, GL_LUMINANCE12_ALPHA12, GL_LUMINANCE16_ALPHA16, GL_INTENSITY, GL_INTENSITY4, GL_INTENSITY8, GL_INTENSITY12, GL_INTENSITY16, GL_R3_G3_B2, GL_RGB, GL_RGB4, GL_RGB5, GL_RGB8, GL_RGB10, GL_RGB12, GL_RGB16, GL_RGBA, GL_RGBA2, GL_RGBA4, GL_RGB5_A1, GL_RGBA8, GL_RGB10_A2, GL_RGBA12, GL_RGBA16, GL_SLUMINANCE, GL_SLUMINANCE8, GL_SLUMINANCE_ALPHA, GL_SLUMINANCE8_ALPHA8, GL_SRGB, GL_SRGB8, GL_SRGB_ALPHA, GL_SRGB8_ALPHA8.

După ce imaginea este citită dintr-un fişier, sau este calculată prin program, **pentru fiecare pixel** există se va rezerva o zonă de memorie internă pentru textura generată, corespunzătoare culorilor de bază şi eventual pentru componenta "alpha". Exemplu de structură în memoria internă pentru formatul **GL_RGBA:**



- **latime, inaltime** - sunt dimensiunile imaginii (puteri ale lui 2)
- **margine** - precizează marginea texturii (0 sau 1)
- **format** - precizeză formatul culorii unui pixel din **datele care generează textura**: **GL_COLOR_INDEX, GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA, GL_RGB, GL_BGR, GL_RGBA, GL_BGRA, GL_LUMINANCE, GL_LUMINANCE_ALPHA**
- **tip** - precizează tipul datelor (valorilor) pentru fiecare din componentele din care se generează textura: **GL_UNSIGNED_BYTE, GL_BYTE, GL_BITMAP, GL_UNSIGNED_SHORT, GL_SHORT, GL_UNSIGNED_INT, GL_INT, GL_FLOAT, GL_UNSIGNED_BYTE_3_3_2, GL_UNSIGNED_BYTE_2_3_3_REV, GL_UNSIGNED_SHORT_5_6_5, GL_UNSIGNED_SHORT_5_6_5_REV, GL_UNSIGNED_SHORT_4_4_4_4, GL_UNSIGNED_SHORT_4_4_4_4_REV, GL_UNSIGNED_SHORT_5_5_5_1, GL_UNSIGNED_SHORT_1_5_5_5_REV, GL_UNSIGNED_INT_8_8_8_8, GL_UNSIGNED_INT_8_8_8_8_REV, GL_UNSIGNED_INT_10_10_10_2, GL_UNSIGNED_INT_2_10_10_10_REV**
- **pixeli** - este adresa şirului de biți din care se generează textura

In continuare se vor descrie trei variante de generare a texturii.

1. Generarea unei texturi prin calcul:

```
public int Gen_Textura(GL gl){
  //genereaza un sir de octeti cu valorile pentru textura
  //la utilizare se poate folosi id-ul texturii, dat de functie
  int i,j;
  int n=32;//64;
  int IDTextura[] = new int[1];
  byte T[] = new byte[n*n*3];  //textura
  int c;
  int k=16;//32;
```

```
    //se construieste vectorul T, cu culorile pixelilor,
    //linie dupa linie ("tabla de sah" cu 2*2 zone)
    //pentru fiecare pixel se rezerva un octet
    for (i=0;i<n;i++){
      for (j=0;j<n;j++){
        c = (int)(i/k)+(int)(j/k);
        if (c==2) c=0;
        c=c*255;   //=>c=0 sau c=255
        T[(j*n+i)*3]   = (byte)c;  //red
        T[(j*n+i)*3+1] = (byte)c;  //green
        T[(j*n+i)*3+2] = (byte)c;  //blue
      }
    }

    //se genereaza o noua textura
    gl.glGenTextures(1, IDTextura, 0);

    //se precizeaza textura curenta
    gl.glBindTexture(gl.GL_TEXTURE_2D, IDTextura[0]);

    //pregateste sirul T ca o succesiune de octeti
    //(tip de data cerut la generarea texturii)
    ByteBuffer valori=ByteBuffer.wrap(T);
    //genereaza o succesiune de octeti din sirul de octeti

    //generarea texturii
    gl.glTexImage2D(gl.GL_TEXTURE_2D,//Textura2D
              0,                     //Niv.de detaliu
              gl.GL_RGB,             //Formatul intern
              n,n,                   //Dim.texturii
              0,                     //Marginea: 0 sau 1
              gl.GL_RGB,             //Formatul culorii
              gl.GL_UNSIGNED_BYTE,   //Tipul datelor pentru pixeli
              valori);               //Adresa tabelului de pixeli
    return IDTextura[0];
  }
```

2. Construirea unei clase care generează o textură folosind un fişier cu o imagine.

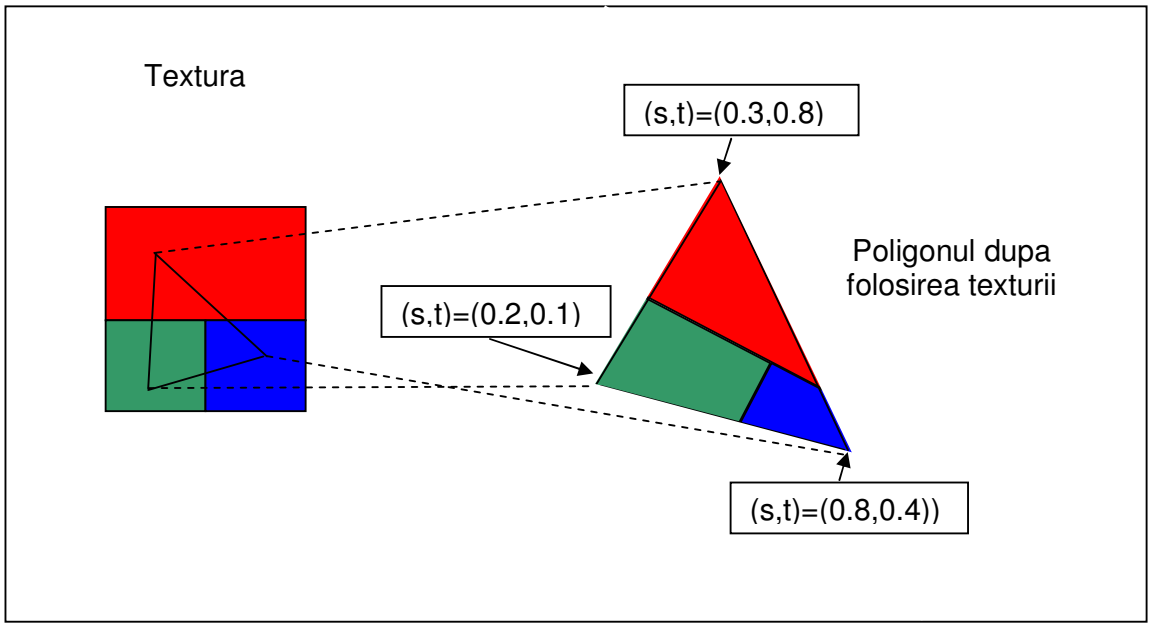
```
import java.awt.*;
import java.awt.image.*;
import java.io.*;
import java.net.*;
import javax.imageio.*;
import javax.media.opengl.*;
import java.nio.*;
import java.awt.geom.*;

public class textura{

  public int id_textura;

  public textura(){
     id_textura=0;
  }
```

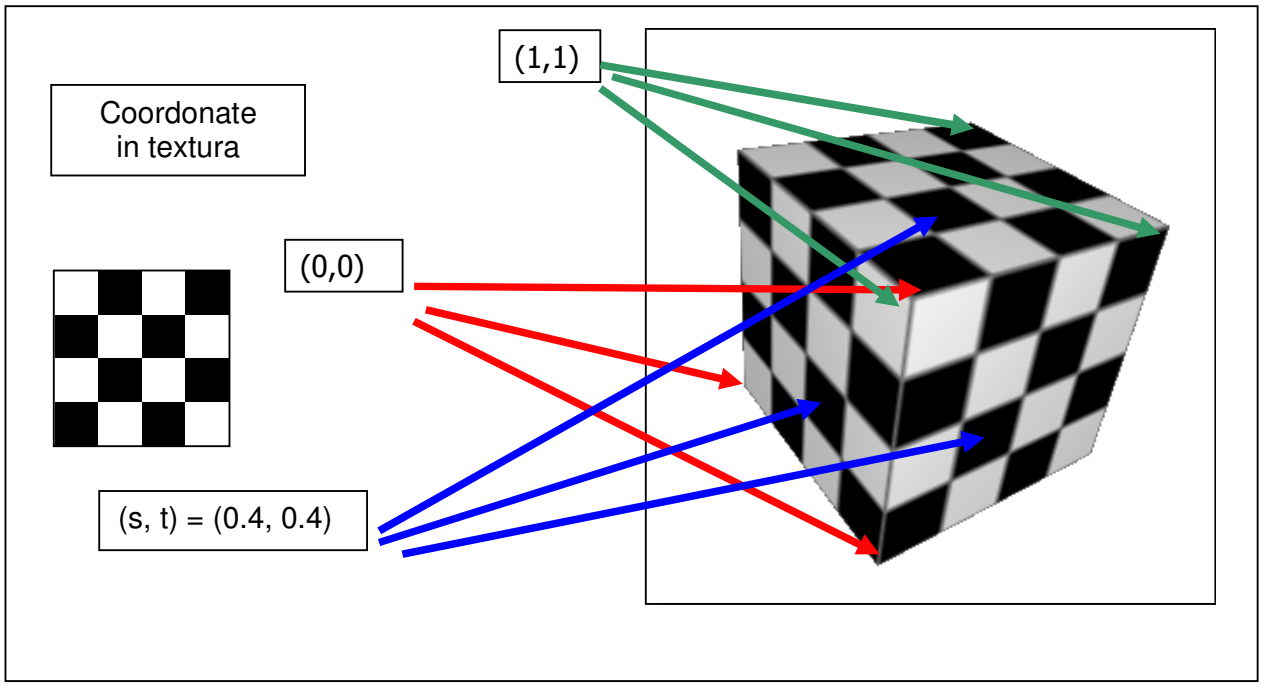
```
public int id(){
   return id_textura;
}

private int UrmPutere(int n){
   int m,p=2;
   while(n>p) p=p*2;
   return p;
}

private static BufferedImage scalare(int w, int h, BufferedImage im)
{
  int hv = im.getHeight();
  int wv = im.getWidth();
  double scX = (double)w / (double)wv;
  double scY = (double)h / (double)hv;
  //schimba dimensiunile imaginii
  AffineTransformOp op = new
       AffineTransformOp(AffineTransform.getScaleInstance(scX,
       scY), null);
  return op.filter(im, null);
}

public int creare(GL gl, String f){
  BufferedImage img;

  try
  {
      URL url = getClass().getResource(f);
      if (url == null)
      {
        throw new RuntimeException("Error " + f);
      }
      img = ImageIO.read(url);
      //citeste imaginea dintr-un fisier
  }
  catch (IOException e)
  {
    throw new RuntimeException(e);
  }

  //determina dimensiunile ca urmatoarele puteri ale lui 2
  int m=UrmPutere(img.getWidth());
  int n=UrmPutere(img.getHeight());
 //scalarea imag.pt. ca fiecare dimens.une sa fie putere a lui 2
  BufferedImage im1=scalare(m,n,img);

  int k = m*n*3;
  byte T[] = new byte[k];
  int i,j;

  Color c;
```

```
    //parcurge imaginea si memoreaza culorile
    for (i=0;i<m;i++){
      for (j=0;j<n;j++){
        c = new Color(im1.getRGB(i,j));
        k=((n-1-j)*m+i)*3;
        T[k] = (byte)c.getRed();
        T[k+1] = (byte)c.getGreen();
        T[k+2] = (byte)c.getBlue();
      }
    }

    //se genereaza o textura
    int adrt[]=new int[1];
    gl.glGenTextures(1, adrt, 0);
    //se precizeaza textura curenta
    gl.glBindTexture(gl.GL_TEXTURE_2D, adrt[0]);
    //pregateste sirul T ca o succesiune de octeti
    ByteBuffer textura=ByteBuffer.wrap(T);
    //Definirea texturii
    gl.glTexImage2D(gl.GL_TEXTURE_2D,     //Textura2D
              0,                    //Niv.de detaliu
              gl.GL_RGB,            //Nr.componentelor de culoare
              m,n,                  //Dim.texturii
              0,                    //Marginea: 0 sau 1
              gl.GL_RGB,            //Formatul culorii
              gl.GL_UNSIGNED_BYTE,  //Tipul datelor pentru pixeli
              textura);             //Adresa tabelului de pixeli
    id_textura=adrt[0];
    return id_textura;
  }
}  //end clasa
```

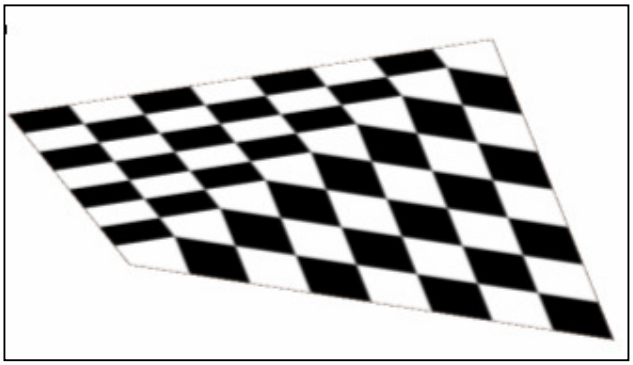
3. Folosirea clasei **TextureIO** din pachetul **com.sun.opengl.util.texture**

Puncte din textura, precizate prin coordonate, se pot suprapune peste puncte din obiectul grafic.
La fiecare vârf generat din fiecare poligon se poate asocia un punct din textură, aşa cum se vede din figurile următoare.





O astfel de aplicare a texturii pentru fiecare primitivă (prin care este desenat obiectul grafic) este o problemă dificilă (aplicarea texturii peste primitive vecine trebuie să "continue" imaginea din care s-a generat textura). O "nepotrivire" poate duce la situația amintită în a doua figură şi reafişată alăturat.

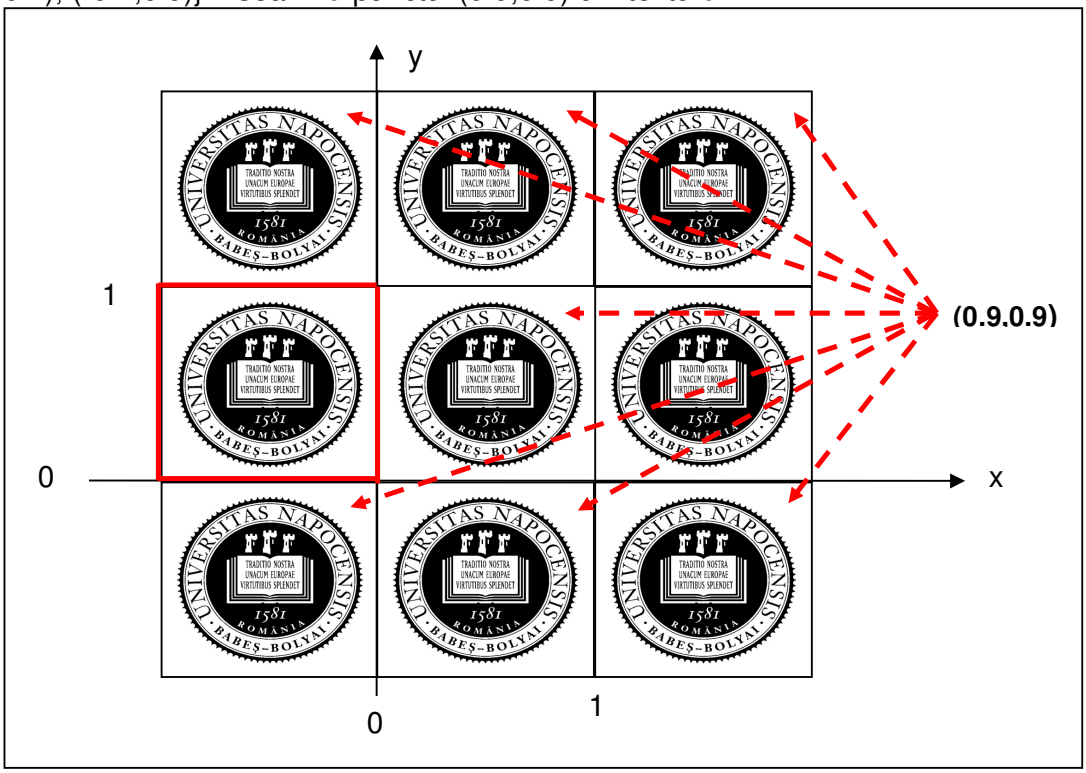


Inainte de a fi aplicată textura peste un obiect, se pot folosi **transformări** ale

acesteia (translatii, rotatii, etc....). Cu **glMatrixMode(G_TEXTURE)** se precizează că transformările care se vor da în continuare (glTranslate, glRotate, etc.) se adresează texturii.

Un **punct curent** din textură se precizează cu: **glTexCoord2f(s,t)** (există şi alte variante pentru comandă). In momentul în care se dă un punct din primitiva de desenare, cu **glVertex**, pentru acest punct **se asociază punctul curent din textură**.

Coordonatele (s,t) sunt în intervalul [0,1]. Dacă aceste coordonate sunt în exteriorul intervalului [0,1], atunci ele se reduc la [0,1] (se ia numai partea fracționară a coordonatelor (s,t)). Se poate considera că patratul [0,1]x[0,1] care conține textura se **multiplică la infinit** în plan. In figura următoare se face o **repetare** a texturii, deci folosirea coordonatelor (s,t) din mulțimea {(0.9,0.9), (1.9,0.9), (0.9,1.9), (1.9,1.9), (0.9,-0.1), (-0.1,-0.1), (-0.1,0.9)} înseamnă punctul (0.9,0.9) din textură.



In afară de această repetare (pe una sau ambele direcții din plan) se poate lua şi varianta de a nu face repetarea şi de lua valorile pixelilor de pe una din laturile texturii, iar utilizarea coordonatelor (1.9,0.9) să fie egală cu (1.0,0.9), deci în acest caz se repetă latura din dreapta texturii definită în [0,1]x[0,1] pentru valorile s>1. Pentru valori s<0 se repetă latura din stânga a texturii. Analog se poate preciza faptul că repetarea texturii să nu se facă pe verticală, deci pentru valorile t<0 sau t>1.

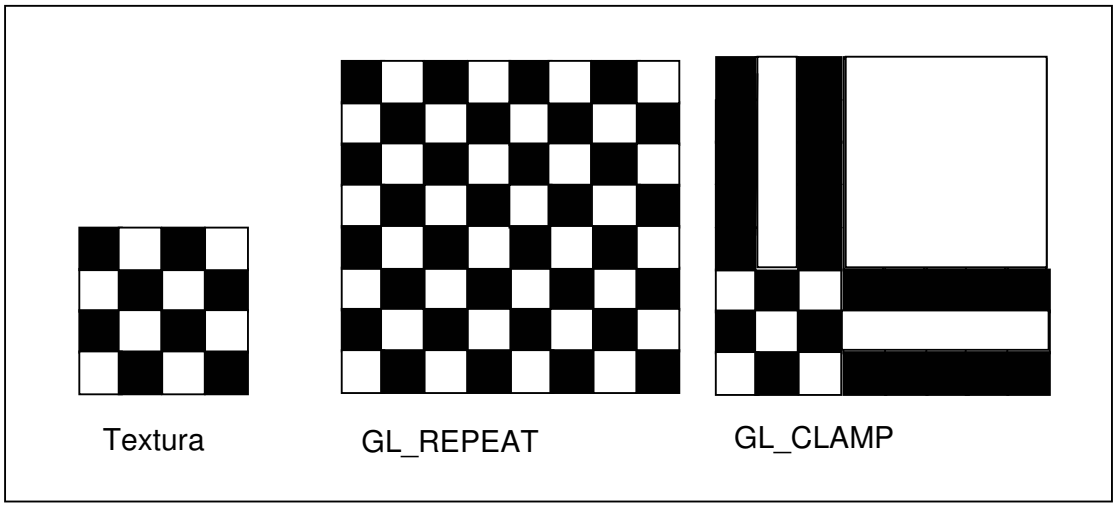
Varianta dorită se precizează cu comanda:

glTexParameteri(GL_TEXTURE_2D, parametru, valoare)

unde "**parametru"** poate fi

`GL_TEXTURE_WRAP_S` sau **`GL_TEXTURE_WRAP_T`**,

corespunzător celor două coordonate din textură (orizontală, verticală). **Valorile** posibile sunt: **GL_CLAMP** (coordonatele se restrâng la intervalul [0,1]) şi **GL_REPEAT** (imaginea se poate multiplica). In figura următoare sunt exemplificate cele două moduri de folosire a texturii amintite mai sus.



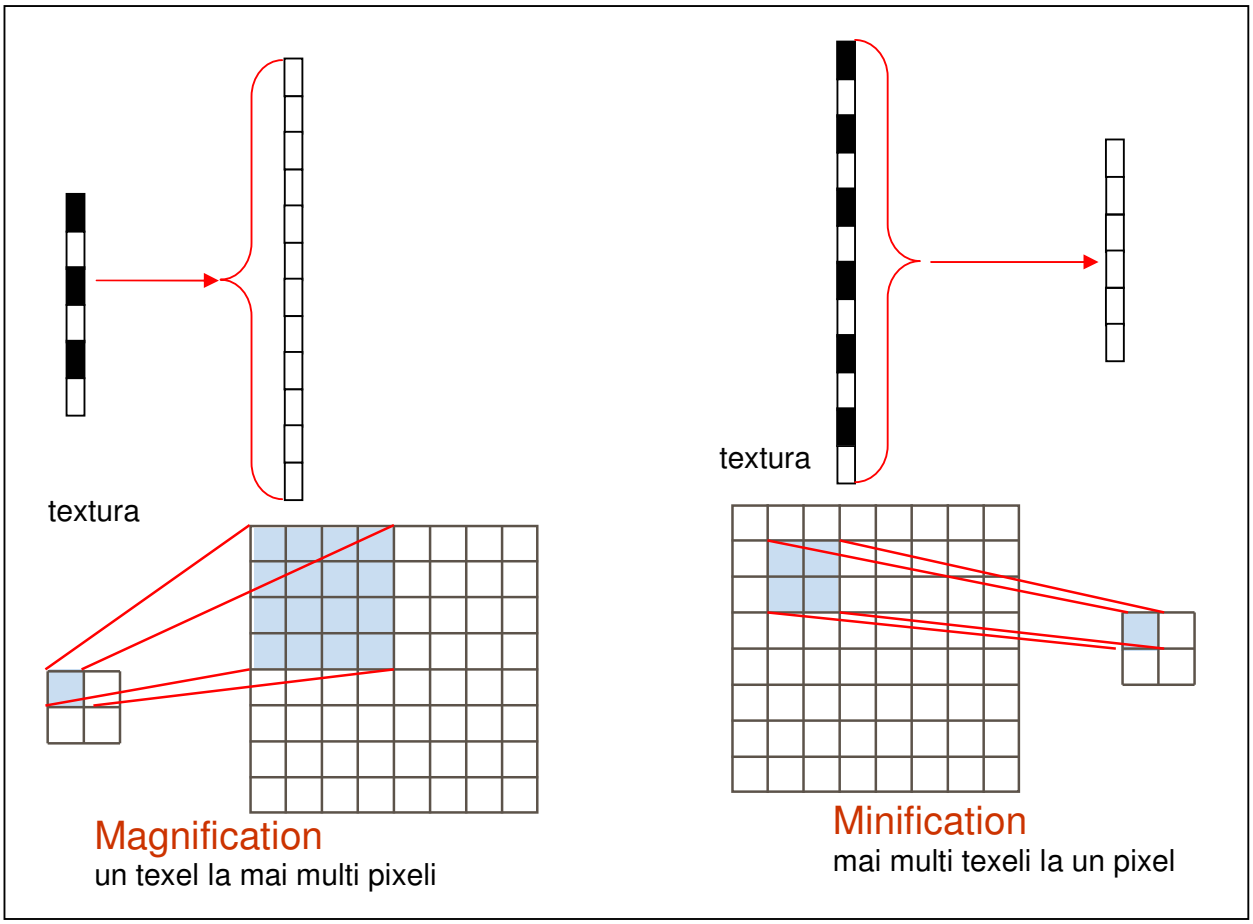
Prin comenzi OpenGL se realizează o corespondență între vârfurile poligoanelor obiectului grafic şi puncte din textură. La **desenarea unui poligon** se **generează** fiecare pixel din transpunerea lui pe ecran. Pentru fiecare pixel din transpunerea poligonului pe ecran va trebui să se determine un pixel din textură. Fie (A) dreptunghiul de pe ecran în care se înscrie transpunerea unui poligon (se obține extensia de pe ecran a poligonului) şi fie (B) dreptunghiul care conține punctele (s,t) folosite la asocierea textelilor pentru vârfurile poligonului.

Pot apare situațiile următoare, pentru fiecare din dimensiunile celor două dreptunghiuri (lățime, înălțime):

- dimensiunea celor două dreptunghiuri sunt egale, deci un texel se transpune într-un pixel
- dimensiunea dreptunghiului (A) este mai mare decât a dreptunghiului (B), deci un texel din textura se aplică (se foloseşte) în mai mulți pixeli pe ecran
- dimensiunea dreptunghiului (A) este mai mică decât a dreptunghiului (B), deci pentru un pixel din (A) sunt necesari mai multi texeli din textură

Având în vedere aceste situații posibile, trebuie precizată o metodă pentru determinarea culorii pixelilor de pe ecran. Pentru aceasta se foloseşte tot comanda:

glTexParameteri(GL_TEXTURE_2D, parametru, valoare)

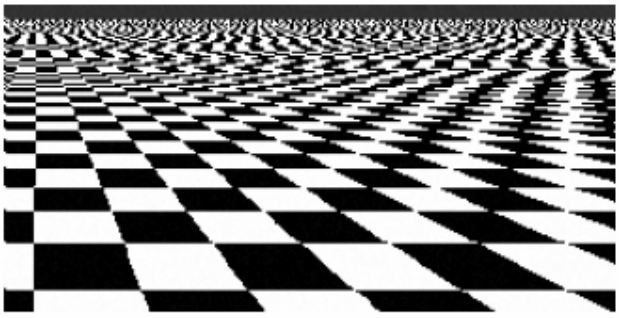


Parametrii care trebuie precizați sunt:
GL_TEXTURE_MAG_FILTER şi **GL_TEXTURE_MIN_FILTER,**
iar valorile pot fi**:**

- **GL_NEAREST** - se ia texelul cel mai apropiat de pixel (se compară centrele zonelor pentru pixel şi texel);
- **GL_LINEAR** - se ia o valoare medie a celor 4 texeli vecini (2x2) (se compară centrele zonelor pentru pixel şi texel).

Exemplu:
glTexParameter(GL_TEXTURE_2D, **GL_TEXTURE_MAG_FILTER**, GL_LINEAR);

Sunt situații unde folosirea unei astfel de texturi genereaza unele "defecte".



Pntru aceasta se permite construirea unei succesiuni de texturi, prima textură este cea folosită mai sus, iar următoarele sunt obținute prin **înjumătățirea dimensiunilor texturii** precedente. La aplicarea texturii peste un poligon se poate alege o variantă prin care să

se aleagă texturile cele mai apropiate de dimensiunea poligonului. Construirea unei astfel de succesiuni de texturi se poate face prin comanda:

```
gluBuild2DMipmaps(int target, int internalFormat, int width, int
    height, int format, int type, Buffer data)
```

unde semnificația parametrilor este precizata la glTexImage2D.

Valorile parametrilor: **GL_TEXTURE_MAG_FILTER,** sau **GL_TEXTURE_MIN_FILTER** pot să fie:

GL_*_MIPMAP_*_
unde *** = NEAREST sau LINEAR**

Plecând de la imaginea inițiala a texturii: 2^m*2^n, se pot face injumătățiri prin scăderea rezoluției: 2^(m-1)*2^(n-1), ..., până se ajunge la o imagine care acoperă cel mai bine pixelul sau la două imagini care incadrează pixelul (după valoarea NEAREST sau LINEAR folosită).