Lucrarea 2

Noţiuni introductive de prelucrarea imaginilor

BREVIAR TEORETIC

2.1 Percepţia imaginilor

Înţelegerea procesului de percepţie a imaginilor de către ochiul uman este foarte importantă pentru dezvoltarea de tehnici de evaluare a calităţii unui sistem sau algoritm de procesare a imaginilor. Informaţia la nivel vizual, conţinută de către o imagine, reprezintă o distribuţie spaţială a unei anumite mărimi, cum ar fi de exemplu luminanţa obiectelor ce compun respectiva imagine. Informaţia percepută de ochiul uman poate fi definită de atribute cum ar fi strălucirea, culoarea sau muchiile unui obiect.

2.1.1 Structura ochiului uman. Formarea imaginii

Ochiul uman are o formă aproape sferică, având un diametru, în medie, de aproximativ 2 cm. Este format din mai multe membrane: *corneea* şi *sclerotica*, ca înveliş exterior, şi *coroida* şi *retina* în interior. Corneea este un ţesut dur şi transparent, pe când sclerotica este o ţesut opac. Coroida se află imediat sub sclerotică şi conţine o reţea de vase de sânge, ce hrănesc ochiul. Este puternic pigmentată pentru a reduce excedentul de lumină ce intră în ochi. Pe cea mai din interior membrană, retina, se formează imaginea, sub influenţa luminii reflectate de obiectele exterioare ochiului.

Retina conţine două tipuri de receptori: *conuri* şi *bastonaşe*. Conurile, în număr de aproximativ 6-7 milioane, servesc la percepţia culorilor. Vederea umană poate percepe detalii foarte fine datorită densităţii mari a acestor receptori, fiecare con fiind legat la o terminaţie nervoasă. Bastonaşele, în număr de aproximativ 75-150 milioane, servesc vederii crepusculare, în condiţii de iluminare slabă. Acestea sunt răspândite pe o arie mare şi conec-

tate la doar câteva terminaţii nervoase, ceea ce are ca efect reducerea considerabilă a perceperii detaliilor din imagine.

Cristalinul joacă rolul de lentilă. El conţine aproximativ 70% apă şi, prin compoziţia lui, permite trecerea a doar 8% din spectrul de radiaţie vizibilă, absorbind în bună măsură şi radiaţiile infraroşii şi ultraviolete.

Imaginea obiectelor se proiectează pe retină, prin cristalin, fiind răsturnată şi având dimensiuni mult mai mici, principiu ce stă la baza aparatului de fotografiat (vezi Figura 2.1).

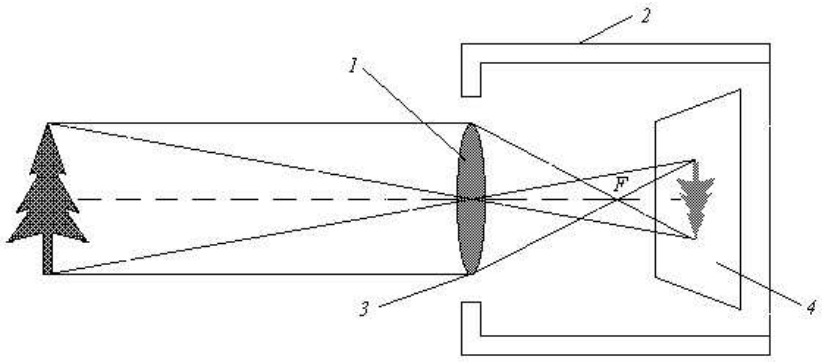

Figura 2.1: Formarea imaginii în aparatul de fotografiat: 1-lentilă, 2-camera obscură, 3-obiectiv, 4-peliculă.

2.1.2 Lumina. Luminanţa. Strălucirea

Lumina este o radiaţie electromagnetică ce stimulează receptorii de la nivelul retinei. Ea se exprimă ca fiind o distribuţie $L(\lambda)$ de energie, unde λ este lungimea de undă a radiaţiei, în cazul nostru vizibilă, cu valori între 350 şi 780 nm. Lumina percepută de la un obiect se poate scrie matematic astfel:

$$I(\lambda) = \rho(\lambda)L(\lambda) \tag{2.1}$$

unde $\rho(\lambda)$ reprezintă măsura în care un obiect reflectă sau transmite energia luminoasă incidentă, a cărei distribuţie este exprimată prin $L(\lambda)$.

Luminanţa sau intensitatea luminoasă a unui obiect cu o distribuţie spaţială a luminii, $I(x, y, \lambda)$, se defineşte astfel:

$$f(x, y) = \int_0^\infty I(x, y, \lambda)V(\lambda)d\lambda \tag{2.2}$$

unde $V(\lambda)$ este *funcţia de eficienţă luminoasă relativă* a sistemului vizual. Pentru ochiul uman, $V(\lambda)$ este o curbă de tip clopot, a cărei caracteristici depind de la o persoană la alta (vezi Figura 2.2).

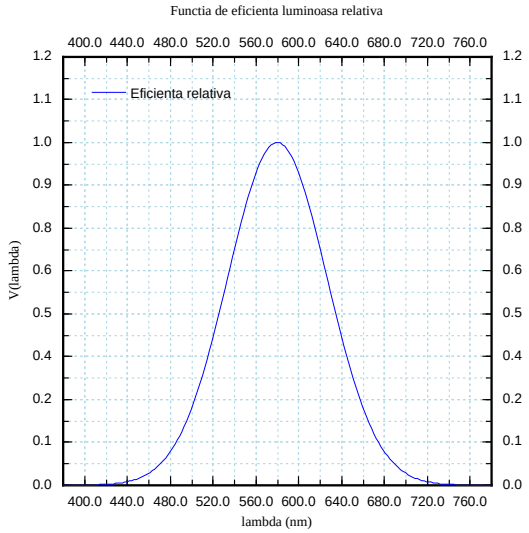

Figura 2.2: Forma tipică a funcţiei de eficienţă luminoasă relativă.

Luminanţa unui obiect este independentă de luminanţa obiectelor din jur.

Strălucirea unui obiect este luminanţa percepută şi depinde de luminanţa mediului ambiant obiectului. Două obiecte aflate într-un acelaşi ambient pot avea aceeaşi luminanţă, dar străluciri diferite.

2.2 Modelul matematic al imaginii

2.2.1 Modelul continuu al imaginii

Matematic imaginile pot fi reprezentate ca o funcţie de două variabile, în spaţiul $L^2(R^2)$, astfel:

- imaginile în tonuri de gri se pot modela cu: $f(x, y) : R^2 \rightarrow R$, caz în care valorile funcţiei f reprezintă valorile luminanţei obiectelor din imagine, în punctele (x, y) ale spaţiului.
- imaginile color se pot modela cu: $f(x, y) : R^2 \rightarrow R^3$, caz în care valorile funcţiei f reprezintă vectori de 3 componente dintr-un spaţiu al culorilor. De exemplu pot fi cele trei componente ale modelului RGB[1].

Spaţiul $L^2(R^2)$ poate fi limitat la un domeniu finit D, ca în Figura 2.3.

[1]RGB = (engl.) Red Green Blue.

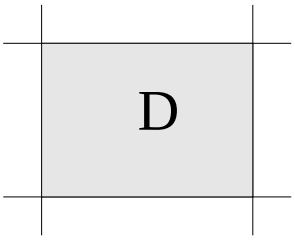

Figura 2.3: Domeniu finit din R^2.

2.2.2 Modelul discret al imaginii

Acesta este modelul utilizat în practică. Funcţia f ia valori discrete, fiind deasemenea definită pe un domeniu de valori discrete, adică:

$$f(k,l) : Z^2 \rightarrow Z^+ \quad sau \quad f(k,l) : Z^2 \rightarrow Z^{+3} \tag{2.3}$$

Trecerea de la domeniul continuu la domeniul discret se face prin eşantionare şi cuantizare.

2.3 Eşantionarea imaginilor

Pentru a putea prelucra cu ajutorul unui calculator o imagine $f(x,y)$, aceasta trebuie discretizată spaţial şi în amplitudine. Discretizarea coordonatelor spaţiale (x,y) poartă numele de *eşantionare*.

Eşantionarea reprezintă procesul de aproximare a unei imagini continue $f(x,y)$ cu o matrice de dimensiune MxN, astfel:

$$f(x,y) \approx \begin{pmatrix} f(0,0) & f(0,1) & ... & f(0,M-1) \\ f(1,0) & f(1,1) & ... & f(1,M-1) \\ . & . & ... & . \\ . & . & ... & . \\ f(N-1,0) & f(N-1,1) & ... & f(N-1,M-1) \end{pmatrix} \tag{2.4}$$

Teorema eşantionării: O imagine $f(x,y)$, având un spectru finit, eşantionată uniform cu o reţea dreptunghiulară de forma celei din Figura 2.4 poate fi refăcută fără eroare din eşantioanele $f(m\delta x, n\delta y)$ cu ajutorul formulei de interpolare:

$$f(x,y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m\delta x, n\delta y) \left(\frac{sin\pi(xu_s - m)}{\pi(xu_s - m)} \right) \left(\frac{sin\pi(xv_s - n)}{\pi(xv_s - n)} \right) \tag{2.5}$$

unde u_s şi v_s reprezintă frecvenţele spaţiale de eşantionare.

Egalitatea dată de teorema eşantionării este valabilă dacă şi numai dacă este respectată condiţia Nyquist, şi anume:

$$\frac{1}{\delta x} = u_s > 2u_0, \qquad \frac{1}{\delta y} = v_s > 2v_0 \tag{2.6}$$

unde u_0 şi v_0 reprezintă frecvenţele spaţiale maxime care apar în imagine.

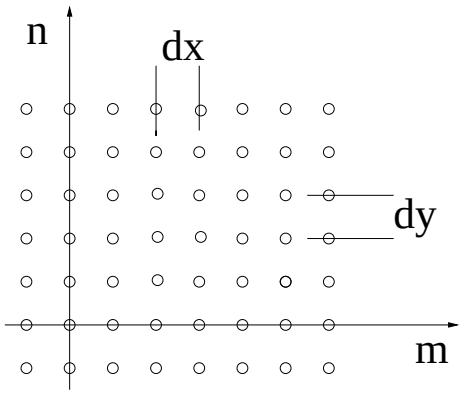

Figura 2.4: Reţea dreptunghiulară de eşantionare.

Cu alte cuvinte, frecvenţele spaţiale de eşantionare trebuie să fie cel puţin dublul frecvenţelor spaţiale maxime conţinute de imagine.

2.4 Cuantizarea imaginilor

Cuantizarea este procesul de discretizare a valorilor funcţiei $f(x, y)$. Aceasta se realizează de obicei cu ajutorul unei funcţii de tip scară, de forma celei din Figura 2.5.

Astfel, tuturor valorilor lui x dintr-un interval li se atribuite valori discrete k. Cuantizarea este un proces însoţit de zgomot, cunoscut sub numele de *eroare de cuantizare*. Cea mai utilizată metodă de cuantizare este cea uniformă, ceea ce înseamna că intervalele funcţiei de cuantizare sunt egale.

2.5 Imaginile digitale

Imaginile astfel discretizate reprezintă structuri bidimensionale de date, denumite imagini digitale. Un element (k, l) al imaginii poartă numele de *pixel*[2].

[2]pixel = (engl. picture + element).

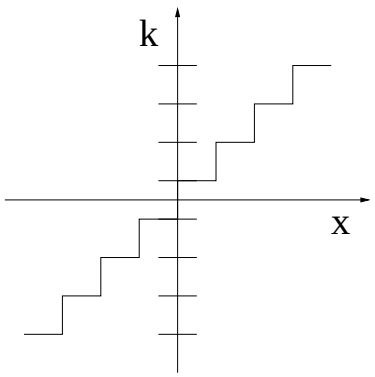

Figura 2.5: Exemplu de funţie de cuantizare.

Imaginile digitale pot fi stocate în memoria sau pe discul unui sistem de procesare şi analiză a imaginilor, în vederea vizualizării sau prelucrării ulterioare. Pe disc imaginile sunt stocate sub forma unor fişiere. Fişierele pot fi de mai multe feluri, în funcţie de formatul în care păstrează datele ce reprezintă imagini: BMP, JPEG, GIF, TIFF, etc.

Cel mai simplu format de fişier imagine este Windows Bitmap (BMP) al firmei Microsoft. Acesta este formatul în care o imagine digitală este stocată practic fără nici un fel de codare sau pierdere de informaţie, cu excepţia reprezentării binare.

DESFĂŞURAREA LUCRĂRII

În continuare este prezentat codul C++ al unei aplicaţi care citeşte o imagine în format BMP şi o afişează într-o fereastră pe ecran. Aplicaţia este dezvoltată utilizând bibliotecile Qt (Linux). Citiţi şi înţelegeţi codul.

Fişierul `aplicatie.h` conţine declararea clasei ImageViewer.

```
#ifndef APLICATIE_H
#define APLICATIE_H

#include <qwidget.h>
#include <qimage.h>
#include <qpainter.h>
#include <qlabel.h>

class QMenuBar;
class QPopupMenu;

class ImageViewer : public QWidget
```

```
{
   Q_OBJECT
   public:
      ImageViewer( QWidget *parent = 0, const char *name = 0,
                   int wFlags = 0 );
      ~ImageViewer();
   bool loadImage( const char *fileName );

   protected:
      void paintEvent( QPaintEvent * );

   private:
      int        conversion_flags;
      int        alloc_context;
      QImage     image;
      QPixmap    pm;
      QMenuBar   *menubar;
      QPopupMenu *file;
      QLabel     *status;
      bool       reconvertImage();

   private slots:
      void  openFile( void );
      void  saveFile( void );
};

#endif // APLICATIE_H
```

Fişierul `aplicatie.cpp` conţine definiţiile funcţiilor membre clasei ImageViewer.

```
#include "aplicatie.h"
#include <qmenubar.h>
#include <qfiledialog.h>
#include <qmessagebox.h>
#include <qpopupmenu.h>
#include <qpainter.h>
#include <qapplication.h>
#include <qwidget.h>

ImageViewer :: ImageViewer( QWidget *parent, const char *name,
    int wFlags ) : QWidget( parent , name , wFlags ),
    conversion_flags( PreferDither ), filename( 0 )
{
   alloc_context = 0;
```

```
   menubar = new QMenuBar( this );
   menubar->setSeparator( QMenuBar :: InWindowsStyle );

   file = new QPopupMenu();
   CHECK_PTR( file );
   menubar->insertItem( "&File" , file );
   file->insertItem( "Open image ( BMP )", this,
         SLOT( openFile() ), CTRL+Key_O );
   file->insertSeparator();
   file->insertItem( "Save as ... ( BMP )", this,
         SLOT( saveFile() ), CTRL+Key_S );
   file->insertSeparator();
   file->insertItem( "Quit", qApp, SLOT(quit()), CTRL+Key_Q );

   status = new QLabel( this );
   status->setFrameStyle( QFrame::WinPanel | QFrame::Sunken );
   status->setFixedHeight( fontMetrics().height() + 4 );

   setMouseTracking( TRUE );
}

ImageViewer :: ~ImageViewer()
{
   if ( alloc_context )
   QColor :: destroyAllocContext( alloc_context );

   delete menubar;
   delete file;
   delete help;
   delete status;
}

void ImageViewer  :: openFile()
{
   QString newfilename = QFileDialog :: getOpenFileName();

   if ( !newfilename.isEmpty() )
   {
      loadImage( newfilename );
      repaint();
   }
}

void ImageViewer :: saveFile( void )
```

```
{
   QImageIO iio;
   QString save_as_file = QFileDialog :: getSaveFileName();

   if( !save_as_file.isEmpty() )
   {
      iio.setImage( image );
      iio.setFileName( save_as_file );
      iio.setFormat( "BMP" );
      iio.write();
   }
}

bool ImageViewer :: loadImage( const char *fileName )
{
   bool ok = FALSE;
   int w, h;

   if( filename )
   {
      QApplication :: setOverrideCursor( waitCursor );
      ok = image.load( filename , 0 );
      pm.convertFromImage( image , conversion_flags );

      if( ok )
      {
         setCaption( filename );
         w = pm.width();
         h = pm.height();
         h += menubar->heightForWidth(w) + status->height();
      }
      else
      {
         pm.resize( 0 , 0 );
         update();
      }

      setFixedWidth( w );
      setFixedHeight( h );
      status->setGeometry( 0, height() - status->height(),
            width(), status->height() );
      QApplication :: restoreOverrideCursor();
   }
   return ok;
```

```
}

void ImageViewer :: paintEvent( QPaintEvent *e )
{
   if( pm.size() != QSize( 0, 0 ) )
   {
      QPainter painter( this );
      painter.setClipRect( e->rect() );
      painter.drawPixmap( 0, menubar->heightForWidth(
            width() ), pm );
   }
}
```

Fişierul `main.cpp` conţine instanţierea unui obiect `QApplication` şi a unuia de tip `ImageViewer`.

```
#include "aplicatie.h"
#include <qapplication.h>
#include <qimage.h>

#ifdef QIMGIO
#include <qimageio.h>
#endif

int main( int argc, char **argv )
{
   QApplication :: setFont( QFont( "Helvetica" , 12 ) );
   QApplication a( argc, argv );

#ifdef QIMGIO
   qInitImageIO();
#endif

  ImageViewer *w = new ImageViewer( 0, "main window",
           QWidget :: WDestructiveClose ) ;
  w->show() ;
  QObject :: connect( qApp, SIGNAL( lastWindowClosed() ),
         qApp, SLOT(quit()) );
  return a.exec() ;
}
```