COURSE NOTES:

IMAGE PROCESSING

George Wolberg

Department of Computer Science
City College of New York
New York, NY 10031
wolberg@cs.ccny.cuny.edu
Introduction to Image Processing

Prof. George Wolberg
Dept. of Computer Science
City College of New York

Course Description

• Intense introduction to image processing.
• Intended for advanced undergraduate and graduate students.
• Topics include:
  - Image enhancement
  - Digital filtering theory, Fourier transforms
  - Image reconstruction, resampling, antialiasing
  - Scanline algorithms, geometric transforms
  - Warping, morphing, and visual effects

Syllabus

<table>
<thead>
<tr>
<th>Week</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction / overview</td>
</tr>
<tr>
<td>2-3</td>
<td>Point operations</td>
</tr>
<tr>
<td>4</td>
<td>Neighborhood operations</td>
</tr>
<tr>
<td>5-6</td>
<td>Fast Fourier transforms (FFT)</td>
</tr>
<tr>
<td>7-8</td>
<td>Sampling theory</td>
</tr>
<tr>
<td>9</td>
<td>Midterm, Image reconstruction</td>
</tr>
<tr>
<td>10</td>
<td>Fast filtering for resampling</td>
</tr>
<tr>
<td>11</td>
<td>Spatial transformations, texture mapping</td>
</tr>
<tr>
<td>12-14</td>
<td>Separable warping algorithms; visual effects</td>
</tr>
</tbody>
</table>

Texts

• Required Text:

• Supplementary Text:

Grading

• The final grade is computed as follows:
  - Midterm exam: 25%
  - Final exam: 25%
  - Homework programming assignments: 50%
• Substantial programming assignments are due every three weeks.
• Proficiency in C/C++ is expected.
• Prereqs: CSc 30100 and CSc 32200

Computer Resources

• SUN Blade workstations (NAC 7/105)
  - Solaris 8 Operating System (UNIX)
  - OpenGL hardware support for fast rendering
  - 24-bit true-color graphics (16.7 million colors)

• Linux Lab (Steinman B41)
  - Red Hat OS (UNIX)
  - OpenGL hardware support for fast rendering
  - 24-bit true-color graphics (16.7 million colors)

• You can also program at home on your PC/laptop under MS Windows or Linux. Download MESA, if necessary, for Linux. Use C/C++ programming language.
Contact Information

- Prof. Wolberg
  - Office hours: After class and by appointment
  - Email: wolberg@cs.ccnycuny.edu
- Teaching Assistant (TA): Hadi Fadaifard
  - Email: cs1025cd@yahoo.com
- See class web page for all class info such as homework and sample source code:
  - www.cs.ccnycuny.edu/~wolberg/cs470 (CCNY)
  - www.cs.ccnycuny.edu/~wolberg/cs4165 (Columbia)

Objectives

- These notes accompany the textbooks:
  - “Digital Image Warping” by George Wolberg
  - “Digital Image Processing” by Gonzalez/Woods
- They form the basis for approximately 14 weeks of lectures.
- Programs in C/C++ will be assigned to reinforce understanding of the material.
  - Four homework assignments
  - Each due in 3 weeks and requiring ~4 programs

What is Image Processing?

Prof. George Wolberg
Dept. of Computer Science
City College or New York

What is Digital Image Processing?

- Computer manipulation of pictures, or images, that have been converted into numeric form.
- Typical operations include:
  - Contrast enhancement
  - Remove blur from an image
  - Smooth out graininess, speckle, or noise
  - Magnify, minify, or rotate an image (image warping)
  - Geometric correction
  - Image compression for efficient storage/transmission

Image Processing Goals

- Image processing is a subclass of signal processing concerned specifically with pictures
- It aims to improve image quality for
  - human perception: subjective
  - computer interpretation: objective
- Compress images for efficient storage/transmission
Related Fields

- Image Processing
- Computer Graphics
- Scene Description
- Computer Vision

Overlap with Related Fields

- Image Processing
- Computer Graphics
- Scene Description
- Computer Vision

Distinctions

- No clear cut boundaries between image processing on the one end and computer vision at the other
- Defining image processing as image-in/image-out does not account for
  - computation of average intensity: image-in / number-out
  - image compression: image-in / coefficients-out
- Nevertheless, image-in / image-out is true most of the time

Image Description

- Image Processing
- Computer Graphics
- Computer Vision
- Artificial Intelligence

Image Processing: 1960-1970

- Geometric correction and image enhancement applied to Ranger 7 pictures of the moon.
- Work conducted at the Jet Propulsion Laboratory.


- Invention of computerized axial tomography (CAT)
- Emergence of medical imaging
- Rapid growth of X-ray imaging for CAT scans, inspection, and astronomy
- LANDSAT earth observation

Image Processing: 1980-1990

- Satellite infrared imaging: LANDSAT, NOAA
- Fast resampling and texture mapping
Image Processing: 1990-2000

- Morphing / visual effects algorithms
- JPEG/MPEG compression, wavelet transforms
- Adobe PhotoShop

Image Processing: 2000-

- Widespread proliferation of fast graphics processing units (GPU) from nVidia and ATI to perform real-time image processing
- Ubiquitous digital cameras, camcorders, and cell phone cameras rely heavily on image processing and compression

Sources of Images

- The principal energy source for images is the electromagnetic energy spectrum.
- EM waves = stream of massless (proton) particles, each traveling in a wavelike pattern at the speed of light. Spectral bands are grouped by energy/photon
  - Gamma rays, X-rays, UV, Visible, Infrared, Microwaves, radio waves
- Other sources: acoustic, ultrasonic, electronic

Gamma-Ray Imaging

- Used in nuclear medicine, astronomy
- Nuclear medicine: patient is injected with radioactive isotope that emits gamma rays as it decays. Images are produced from emissions collected by detectors.

X-Ray Imaging

- Oldest source of EM radiation for imaging
- Used for CAT scans
- Used for angiograms where X-ray contrast medium is injected through catheter to enhance contrast at site to be studied.
- Industrial inspection

Ultraviolet Imaging

- Used for lithography, industrial inspection, fluorescence microscopy, lasers, biological imaging, and astronomy
- Photon of UV light collides with electron of fluorescent material to elevate its energy. Then, its energy falls and it emits red light.
Visible and Infrared Imaging (1)

- Used for astronomy, light microscopy, remote sensing

Visible and Infrared Imaging (2)

- Industrial inspection
  - inspect for missing parts
  - missing pills
  - unacceptable bottle fill
  - unacceptable air pockets
  - anomalies in cereal color
  - incorrectly manufactured replacement lens for eyes

Microwave Imaging

- Radar is dominant application
- Microwave pulses are sent out to illuminate scene
- Antenna receives reflected microwave energy

Radio-Band Imaging

- Magnetic resonance imaging (MRI):
  - places patient in powerful magnet
  - passes radio waves through body in short pulses
  - each pulse causes a responding pulse of radio waves to be emitted by patient’s tissues
  - Location and strength of signal is recorded to form image

Images Covering EM Spectrum

Non-EM modality: Ultrasound

- Used in geological exploration, industry, medicine:
  - transmit high-freq (1-5 MHz) sound pulses into body
  - record reflected waves
  - calculate distance from probe to tissue/organ using the speed of sound (1540 m/s) and time of echo’s return
  - display distance and intensities of echoes as a 2D image
**Non-EM modality:** Scanning Electron Microscope

- Stream of electrons is accelerated toward specimen using a positive electrical potential
- Stream is focused using metal apertures and magnetic lenses into a thin beam
- Scan beam; record interaction of beam and sample at each location (dot on phosphor screen)

**Visible Spectrum**

- Thin slice of the full electromagnetic spectrum

**Objectives**

- In this lecture we discuss:
  - Structure of human eye
  - Mechanics of human visual system (HVS)
  - Brightness adaptation and discrimination
  - Perceived brightness and simultaneous contrast

**Human and Computer Vision**

- We observe and evaluate images with our visual system
- We must therefore understand the functioning of the human visual system and its capabilities for brightness adaptation and discrimination:
  - What intensity differences can we distinguish?
  - What is the spatial resolution of our eye?
  - How accurately do we estimate distances and areas?
  - How do we sense colors?
  - By which features can we detect/distinguish objects?

**Examples**

- Parallel lines: <5\% variation in length
- Circles: <10\% variation in radii
- Vertical line falsely appears longer
- Upper line falsely appears longer
Structure of the Human Eye

- Shape is nearly spherical
- Average diameter = 20mm
- Three membranes:
  - Cornea and Sclera
  - Choroid
  - Retina

Structure of the Human Eye: Cornea and Sclera

- Cornea
  - Tough, transparent tissue that covers the anterior surface of the eye
- Sclera
  - Opaque membrane that encloses the remainder of the optical globe

Structure of the Human Eye: Choroid

- Choroid
  - Lies below the sclera
  - Contains network of blood vessels that serve as the major source of nutrition to the eye.
  - Choroid coat is heavily pigmented and hence helps to reduce the amount of extraneous light entering the eye and the backscatter within the optical globe

Lens and Retina

- Lens
  - Both infrared and ultraviolet light are absorbed appreciably by proteins within the lens structure and, in excessive amounts, can cause damage to the eye
- Retina
  - Innermost membrane of the eye which lines the inside of the wall’s entire posterior portion. When the eye is properly focused, light from an object outside the eye is imaged on the retina.

Receptors

- Two classes of light receptors on retina: cones and rods
- Cones
  - 6-7 million cones lie in central portion of the retina, called the fovea.
  - Highly sensitive to color and bright light.
  - Resolve fine detail since each is connected to its own nerve end.
  - Cone vision is called photopic or bright-light vision.
- Rods
  - 75-150 million rods distributed over the retina surface.
  - Reduced amount of detail discernable since several rods are connected to a single nerve end.
  - Serves to give a general, overall picture of the field of view.
  - Sensitive to low levels of illumination.
  - Rod vision is called scotopic or dim-light vision.

Distribution of Cones and Rods

- Blind spot: no receptors in region of emergence of optic nerve.
- Distribution of receptors is radially symmetric about the fovea.
- Cones are most dense in the center of the retina (e.g., fovea)
- Rods increase in density from the center out to 20° and then decrease.
Brightness Adaptation (1)

- The eye’s ability to discriminate between intensities is important.
- Experimental evidence suggests that subjective brightness (perceived) is a logarithmic function of light incident on eye.

![Graph showing subjective brightness vs. log intensity](image)

- Notice approximately linear response in log-scale below.
- Wide range of intensity levels to which HVS can adapt: from scotopic threshold to glare limit (on the order of 10^{10}).
- Range of subjective brightness that eye can perceive when adapted to level B_a.

Brightness Adaptation (2)

- Essential point: the HVS cannot operate over such a large range simultaneously.
- It accomplishes this large variation by changes in its overall sensitivity: brightness adaptation.
- The total range of distinct intensity levels it can discriminate simultaneously is rather small when compared with the total adaptation range.
- For any given set of conditions, the current sensitivity level of the HVS is called the brightness adaptation level (B_a in figure).

Brightness Discrimination (1)

- The ability of the eye to discriminate between intensity changes at any adaptation level is of considerable interest.
- Let I be the intensity of a large uniform area that covers the entire field of view.
- Let ΔI be the change in object brightness required to just distinguish the object from the background.
- Good brightness discrimination: ΔI / I is small.
- Bad brightness discrimination: ΔI / I is large.
- ΔI / I is called Weber’s ratio.

Brightness Discrimination (2)

- Brightness discrimination is poor at low levels of illumination, where vision is carried out by rods. Notice Weber’s ratio is large.
- Brightness discrimination improves at high levels of illumination, where vision is carried out by cones. Notice Weber’s ratio is small.

Choice of Grayscales (1)

- Let I take on 256 different intensities:
  - 0 ≤ I_j ≤ 1 for j = 0,1,…,255.
- Which levels we use?
  - Use eye characteristics: sensitive to ratios of intensity levels rather than to absolute values (Weber’s law: ΔB/B = constant)
  - For example, we perceive intensities .10 and .11 as differing just as much as intensities .50 and .55.

Choice of Grayscales (2)

- Levels should be spaced logarithmically rather than linearly to achieve equal steps in brightness:
  - I_0, I_1 = rI_0, I_2 = rI_1, I_3 = rI_2, …., I_{255} = r^{255}I_0 = I_0
  - In general, for n+1 intensities:
    - r = (1/I_0)^{1/n},
    - I_j = I_0 (r^{j-1}) for 0 ≤ j ≤ n
Choice of Grayscales (3)

- Example: let $n=3$ and $I_0=1/8$:
  - $r = 2$
  - $I_2 = (1/8)^{(2/3)}$
  - $I_1 = (1/8)^{(1/3)} = \frac{1}{4}$
  - $I_2 = (1/8)^{(1/3)} = \frac{1}{2}$
  - $I_3 = (1/8)^{(0/3)} = 1$
- For CRTs, $1/200 < I_0 < 1/40$.
- $I_0 \neq 0$ because of light reflection from the phosphor within the CRT.
- Linear grayscale is close to logarithmic for large number of graylevels (256).

Perceived Brightness

- Perceived brightness is not a simple function of intensity.
- The HVS tends to over/undershoot around intensity discontinuities.
- The scalloped brightness bands shown below are called Mach bands, after Ernst Mach who described this phenomenon in 1865.

Simultaneous Contrast (1)

- A region’s perceived brightness does not depend simply on its intensity. It is also related to the surrounding background.

Simultaneous Contrast (2)

- An example with colored squares.

Projectors

- Why are projection screens white?
  - Reflects all colors equally well
- Since projected light cannot be negative, how are black areas produced?
  - Exploit simultaneous contrast
  - The bright area surrounding a dimly lit point makes that point appear darker

Visual Illusions (1)
Visual Illusions (2)

- Rotating snake illusion
- Rotation occurs in relation to eye movement
- Effect vanishes on steady fixation
- Illusion does not depend on color
- Rotation direction depends on the polarity of the luminance steps
- Asymmetric luminance steps are required to trigger motion detectors

Digital Image Fundamentals

Prof. George Wolberg
Dept. of Computer Science
City College of New York

Objectives

- In this lecture we discuss:
  - Image acquisition
  - Sampling and quantization
  - Spatial and graylevel resolution

Sensor Arrangements

- Three principal sensor arrangements:
  - Single, line, and array

Single Sensor

- Photodiode: constructed of silicon materials whose output voltage waveform is proportional to light.
- To generate a 2D image using a single sensor, there must be relative displacements in the horizontal and vertical directions between the sensor and the area to be imaged.
- Microdensitometers: mechanical digitizers that use a flat bed with the single sensor moving in two linear directions.

Sensor Strips

- In-line arrangement of sensors in the form of a sensor strip.
- The strip provides imaging elements in one direction.
- Motion perpendicular to strip images in the other direction.
- Used in flat bed scanners, with 4000 or more in-line sensors.
Sensor Arrays

- Individual sensors are arranged in the form of a 2D array.
- Used in digital cameras and camcorders.
- Entire image formed at once; no motion necessary.

Signals

- A signal is a function that conveys information
  - 1D signal: \( f(x) \) waveform
  - 2D signal: \( f(x,y) \) image
  - 3D signal: \( f(x,y,z) \) volumetric data
  - 4D signal: \( f(x,y,z,t) \) snapshots of volumetric data
  - 5D signal: \( f(x,y,z,t) \) animation (spatiotemporal volume)
- The dimension of the signal is equal to its number of indices.
- In this course, we focus on 2D images: \( f(x,y) \)
- Efficient implementation often calls for 1D row or column processing. That is, process the rows independently and then process the columns of the resulting image.

Digital Image

- Image produced as an array (the raster) of picture elements (pixels or pels) in the frame buffer.

Image Classification (1)

- Images can be classified by whether they are defined over all points in the spatial domain and whether their image values have finite or infinite precision.
- If the position variables \((x,y)\) are continuous, then the function is defined over all points in the spatial domain.
- If \((x,y)\) is discrete, then the function can be sampled at only a finite set of points, i.e., the set of integers.
- The value that the function returns can also be classified by its precision, independently of \(x\) and \(y\).

Image Classification (2)

- Quantization refers to the mapping of real numbers onto a finite set: a many-to-one mapping.
- Akin to casting from double precision to an integer.

<table>
<thead>
<tr>
<th>Space</th>
<th>Image Values</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>continuous</td>
<td>continuous</td>
<td>analog (continuous) image</td>
</tr>
<tr>
<td>continuous</td>
<td>discrete</td>
<td>intensity quantization</td>
</tr>
<tr>
<td>discrete</td>
<td>continuous</td>
<td>spatial quantization</td>
</tr>
<tr>
<td>discrete</td>
<td>discrete</td>
<td>digital (discrete) image</td>
</tr>
</tbody>
</table>

Image Formation

- The values of an acquired image are always positive. There are no negative intensities: \( 0 < f(x,y) < \infty \)
- Continuous function \( f(x,y) = i(x,y) \cdot r(x,y) \), where \( 0 < i(x,y) \leq \infty \) is the illumination
  \( 0 < r(x,y) < 1 \) is the reflectance of the object
- \( r(x,y) = 0 \) is total absorption and \( r(x,y) = 1 \) is total reflectance.
- Replace \( r(x,y) \) with transmissivity term \( t(x,y) \) for chest X-ray.
Typical Values of Illumination and Reflectance

- The following \((x,y)\) illumination values are typical (in lumens/m²):
  - Illumination of sun on Earth on a clear day: 90,000
  - Illumination of sun on Earth on a cloudy day: 10,000
  - Illumination of moon on Earth on a clear night: 0.1
  - Illumination level in a commercial office: 1000
  - Illumination level of video projectors: 1000-1500

- The following \((x,y)\) reflectance values are typical:
  - Black velvet: 0.01
  - Stainless steel: 0.65
  - Flat-white wall paint: 0.80
  - Silver-plated metal: 0.90
  - Snow: 0.93

Graylevels

- The intensity of a monochrome image at any coordinate \((x,y)\) is called graylevel \(L\), where \(L_{\text{min}} \leq L \leq L_{\text{max}}\)
- The office illumination example indicates that we may expect \(L_{\text{min}} = 0.01 \times 1000 = 10\) (virtual black)
- \(L_{\text{max}} = 1 \times 1000 = 1000\) (white)
- Interval \([L_{\text{min}}, L_{\text{max}}]\) is called the grayscale.
- In practice, the interval is shifted to the \([0, 255]\) range so that intensity can be represented in one byte (unsigned char).
- 0 is black, 255 is white, and all intermediate values are different shades of gray varying from black to white.

Generating a Digital Image

- Sample and quantize continuous input image.

Image Sampling and Quantization

- Sampling: digitize (discretize) spatial coordinate \((x,y)\)
- Quantization: digitize intensity level \(L\)

Effects of Varying Sampling Rate (1)

- Subsampling was performed by dropping rows and columns.
- The number of gray levels was kept constant at 256.

Effects of Varying Sampling Rate (2)

- Size differences make it difficult to see effects of subsampling.
Spatial Resolution

- Defined as the smallest discernable detail in an image.
- Widely used definition: smallest number of discernable line pairs per unit distance (100 line pairs/millimeter).
- A line pair consists of one line and its adjacent space.
- When an actual measure of physical resolution is not necessary, it is common to refer to an MxN image as having spatial resolution of MxN pixels.

Graylevel Resolution

- Defined as the smallest discernable change in graylevel.
- Highly subjective process.
- The number of graylevels is usually a power of two:
  - k bits of resolution yields $2^k$ graylevels.
  - When $k=8$, there are 256 graylevels — most typical case
- Black-and-white television uses $k=6$, or 64 graylevels.

Effects of Varying Graylevels (1)

- Number of graylevels reduced by dropping bits from $k=8$ to $k=1$
- Spatial resolution remains constant.

Effects of Varying Graylevels (2)

- Notice false contouring in coarsely quantized images.
- Appear as fine ridgelike structures in areas of smooth gray levels.

Storage Requirements

- Consider an N x N image having k bits per pixel.
- Color (RGB) images require three times the storage (assuming no compression).

| Table 2.1 Number of storage bits for various values of N and k |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| N x N | $2^N$ | $2^{N+1}$ | $2^{N+2}$ | $2^{N+3}$ | $2^{N+4}$ | $2^{N+5}$ | $2^{N+6}$ | $2^{N+7}$ |
| 32 x 32 | 4,096 | 8,192 | 16,384 | 32,768 | 65,536 | 131,072 | 262,144 | 524,288 |
| 64 x 64 | 8,192 | 16,384 | 32,768 | 65,536 | 131,072 | 262,144 | 524,288 | 1,048,576 |
| 128 x 128 | 16,384 | 32,768 | 65,536 | 131,072 | 262,144 | 524,288 | 1,048,576 | 2,097,152 |
| 256 x 256 | 32,768 | 65,536 | 131,072 | 262,144 | 524,288 | 1,048,576 | 2,097,152 | 4,194,304 |
| 512 x 512 | 65,536 | 131,072 | 262,144 | 524,288 | 1,048,576 | 2,097,152 | 4,194,304 | 8,388,608 |
| 1024 x 1024 | 131,072 | 262,144 | 524,288 | 1,048,576 | 2,097,152 | 4,194,304 | 8,388,608 | 16,777,216 |
| 2048 x 2048 | 262,144 | 524,288 | 1,048,576 | 2,097,152 | 4,194,304 | 8,388,608 | 16,777,216 | 33,554,432 |

Large Space of Images

- Any image can be downsampled and represented in a few bits/pixel for use on small coarse displays (PDA).
- How many unique images can be displayed on an N x N k-bit display?
  - $2^k$ possible values at each pixel
  - $N^2$ pixels
  - Total: $(2^k)^2$
- This total is huge even for k=1 and N=8:
  - $18,446,744,073,709,551,616$ — $2^{64}$
- It would take 19,498,080,578 years to view this if it were laid out on video at 30 frames/sec.
Point Operations

Prof. George Wolberg
Dept. of Computer Science
City College or New York

Objectives

- In this lecture we describe point operations commonly used in image processing:
  - Thresholding
  - Quantization (aka posterization)
  - Gamma correction
  - Contrast/brightness manipulation
  - Histogram equalization/matching

Point Operations

- Output pixels are a function of only one input point:
  \[ g(x,y) = T[f(x,y)] \]
- Transformation \( T \) is implemented with a lookup table:
  - An input value indexes into a table and the data stored there is copied to the corresponding output position.
  - The LUT for an 8-bit image has 256 entries.

\[
\begin{array}{cccccccc}
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
\end{array}
\]

\[ g(x,y) = T[f(x,y)] \]

Input: \( f(x,y) \)  
Output: \( g(x,y) \)

Graylevel Transformation \( T \)

- Contrast enhancement:
  - Darkens levels below \( m \)
  - Brightens levels above \( m \)

- Thresholding:
  - Replace values below \( m \) to black (0)
  - Replace values above \( m \) to white (255)

Lookup Table: Threshold

- Init LUT with samples taken from thresholding function \( T \)

Lookup Table: Quantization

- Init LUT with samples taken from quantization function \( T \)
Threshold Program

- Straightforward implementation:

```c
// iterate over all pixels
for(i=0; i<total; i++) {
    if(in[i] < thr) out[i] = BLACK;
    else out[i] = WHITE;
}
```

- Better approach: exploit LUT to avoid total comparisons:

```c
// init lookup tables
for(i=0; i<thr; i++) lut[i] = BLACK;
for(; i<MXGRAY; i++) lut[i] = WHITE;
// iterate over all pixels
for(i=0; i<total; i++) out[i] = lut[in[i]];
```

Quantization Program

- Straightforward implementation:

```c
// iterate over all pixels
for(i=0; i<total; i++) {
    scale = MXGRAY / levels;
    out[i] = scale * (int)(in[i]/scale);
}
```

- Better approach: exploit LUT to avoid total multi/division:

```c
// init lookup tables
scale = MXGRAY / levels;
for(i=0; i<MXGRAY; i++)
    lut[i] = scale * (int)(i/scale);
// iterate over all pixels
for(i=0; i<total; i++) out[i] = lut[in[i]];
```

Quantization Artifacts

- The false contours associated with quantization are most noticeable in smooth areas.
- They are obscured in highly textured regions.

Dither Signal

- Reduce quantization error by adding uniformly distributed white noise (dither signal) to the input image prior to quantization.
- Dither hides objectional artifacts.
- To each pixel of the image, add a random number in the range \([-m, m]\), where \(m\) is \(\text{MXGRAY}/\text{quantization-levels}\).

Comparison

1 bpp 2 bpp 3 bpp 4 bpp

Enhancement

- Point operations are used to enhance an image.
- Processed image should be more suitable than the original image for a specific application.
- Suitability is application-dependent.
- A method which is quite useful for enhancing one image may not necessarily be the best approach for enhancing another image.
- Very subjective
Two Enhancement Domains

- **Spatial Domain:** (image plane)
  - Techniques are based on direct manipulation of pixels in an image
- **Frequency Domain:**
  - Techniques are based on modifying the Fourier transform of an image
- There are some enhancement techniques based on various combinations of methods from these two categories.

Enhanced Images

- **For human vision**
  - The visual evaluation of image quality is a highly subjective process.
  - It is hard to standardize the definition of a good image.
- **For machine perception**
  - The evaluation task is easier.
  - A good image is one which gives the best machine recognition results.
- A certain amount of trial and error usually is required before a particular image enhancement approach is selected.

Three Basic Graylevel Transformation Functions

- **Linear function**
  - Negative and identity transformations
- **Logarithmic function**
  - Log and inverse-log transformations
- **Power-law function**
  - $n$th power and $n$th root transformations

Image Negatives

- **Negative transformation**: $s = (L-1) - r$
- Reverses the intensity levels of an image.
- Suitable for enhancing white or gray detail embedded in dark regions of an image, especially when black area is large.

Log Transformations

- $s = c \log (1+r)$
  - $c$ is constant and $r \geq 0$
  - Log curve maps a narrow range of low graylevels in input image into a wider range of output levels.
  - Expands range of dark image pixels while shrinking bright range.
  - Inverse log expands range of bright image pixels while shrinking dark range.

Example of Logarithm Image

- Fourier spectrum image can have intensity range from 0 to $10^6$ or higher.
- Log transform lets us see the detail dominated by large intensity peak.
  - Must now display $[0,6]$ range instead of $[0,10^6]$ range.
  - Rescale $[0,6]$ to the $[0,255]$ range.
Power-Law Transformations

\[ s = cr^\gamma \]

- \( c \) and \( \gamma \) are positive constants
- Power-law curves with fractional values of \( \gamma \) map a narrow range of dark input values into a wider range of output values, with the opposite being true for higher values of input levels.
- \( c = \gamma = 1 \) is identity function

Gamma Correction

- Cathode ray tube (CRT) devices have an intensity-to-voltage response that is a power function, with \( \gamma \) varying from 1.8 to 2.5
- This darkens the picture.
- Gamma correction is done by preprocessing the image before inputting it to the monitor.

Example: MRI

(a) Dark MRI. Expand graylevel range for contrast manipulation
  (b) \( \gamma = 0.6 \), \( c=1 \)
  (c) \( \gamma = 0.4 \) (best result)
  (d) \( \gamma = 0.3 \) (limit of acceptability)

When \( \gamma \) is reduced too much, the image begins to reduce contrast to the point where it starts to have a “washed-out” look, especially in the background.

Example: Aerial Image

Washed-out image. Shrink graylevel range
  (b) \( \gamma = 3.0 \) (suitable)
  (c) \( \gamma = 4.0 \) (suitable)
  (d) \( \gamma = 5.0 \) (High contrast; the image has areas that are too dark; some detail is lost)

Piecewise-Linear Transformation Functions

- Advantage:
  - The form of piecewise functions can be arbitrarily complex
- Disadvantage:
  - Their specification requires considerably more user input

Contrast Stretching

- Low contrast may be due to poor illumination, a lack of dynamic range in the imaging sensor, or even a wrong setting of a lens aperture during acquisition.
- Applied contrast stretching: \( (r_1, s_1) = (r_{min}, 0) \) and \( (r_2, s_2) = (r_{max}, L-1) \)
Graylevel Slicing

Bit-plane slicing

• Highlighting the contribution made to total image appearance by specific bits
• Suppose each pixel is represented by 8 bits
• Higher-order bits contain the majority of the visually significant data
• Useful for analyzing the relative importance played by each bit of the image

Example

• The (binary) image for bit-plane 7 can be obtained by processing the input image with a thresholding graylevel transformation.
  - Map all levels between 0 and 127 to 0
  - Map all levels between 129 and 255 to 255

An 8-bit fractal image

8-Bit Planes

Hardware LUTs

• All point operations can be implemented by LUTs.
• Hardware LUTs operate on the data as it is being displayed.
• It's an efficient means of applying transformations because changing display characteristics only requires loading a new table and not the entire image.
• For a 1024x1024 8-bit image, this translates to 256 entries instead of one million.
• LUTs do not alter the contents of original image (nondestructive).

Histogram

• A histogram of a digital image with gray levels in the range [0, L-1] is a discrete function \( h(r_k) = n_k \)
  - \( r_k \): the \( k^{th} \) gray level
  - \( n_k \): the number of pixels in the image having gray level \( r_k \)
• The sum of all histogram entries is equal to the total number of pixels in the image.
Histogram Example

5x5 image

<table>
<thead>
<tr>
<th>Pixel Value</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>20</td>
</tr>
</tbody>
</table>

Plot of the Histogram

Histogram evaluation:
for(i=0; i<MXGRAY; i++) H[i] = 0;
for(i=0; i<total; i++) H[im[i]]++;

Normalized Histogram

- Divide each histogram entry at gray level \( r_k \) by the total number of pixels in the image, \( n \)
  \[ p( r_k ) = \frac{n_k}{n} \]
- \( p( r_k ) \) gives an estimate of the probability of occurrence of gray level \( r_k \)
- The sum of all components of a normalized histogram is equal to 1.

Histogram Processing

- Basic for numerous spatial domain processing techniques.
- Used effectively for image enhancement:
  - Histogram stretching
  - Histogram equalization
  - Histogram matching
- Information inherent in histograms also is useful in image compression and segmentation.

Example: Dark/Bright Images

- Dark image: Components of histogram are concentrated on the low side of the gray scale.
- Bright image: Components of histogram are concentrated on the high side of the gray scale.

Example: Low/High Contrast Images

- Low-contrast image: histogram is narrow and centered toward the middle of the gray scale
- High-contrast image: histogram covers broad range of the gray scale and the distribution of pixels is not too far from uniform, with very few vertical lines being much higher than the others

Histogram Stretching

1) Slide histogram down to 0
2) Normalize histogram to \([0,1]\) range
3) Rescale to \([0,255]\) range

\[ g = \frac{255(f - \text{MIN})}{\text{MAX} - \text{MIN}} \]
Example (1)

- Improve effectiveness of histogram stretching by clipping intensities first

Example (2)

- Improve effectiveness of histogram stretching by clipping intensities first

Histogram Equalization (1)

- Produce image with flat histogram
- All graylevels are equally likely
- Appropriate for images with wide range of graylevels
- Inappropriate for images with few graylevels (see below)

Histogram Equalization (2)

Objective: we want a uniform histogram.
Rationale: maximize image entropy.

\[ h_i(v_{avg}) = \text{constant} = \frac{\text{total}}{MXGRAY} = h_{avg} \]

\[ c_i(v_{avg}) = (v_{avg} + i) \cdot h_{avg} \]

This is a special case of histogram matching.
Perfectly flat histogram: \( H[v] = \text{total}/MXGRAY \) for \( 0 \leq i < MXGRAY \).
If \( H[v] = k \cdot h_{avg} \) then \( v \) must be mapped onto \( k \) different levels, from \( v_1 \) to \( v_k \). This is a one-to-many mapping.

Histogram Equalization Mappings

Rule 1: Always map \( v \) onto \( (v_1 + v_k)/2 \). (This does not result in a flat histogram, but one where brightness levels are spaced apart).
Rule 2: Assign at random one of the levels in \([v_1, v_k]\).
This can result in a loss of contrast if the original histogram had two distinct peaks that were far apart (i.e., an image of text).
Rule 3: Examine neighborhood of pixel, and assign it a level from \([v_1, v_k]\) which is closest to neighborhood average. This can result in blurriness; more complex.
Rule (1) creates a lookup table beforehand.
Rules (2) and (3) are runtime operations.

Example (1)
The quality is not improved much because the original image already has a wide graylevel scale.

### Implementation (1)

<table>
<thead>
<tr>
<th>No. of pixels</th>
<th>Gray level</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

4x4 image
Gray scale = [0,9]

### Implementation (2)

<table>
<thead>
<tr>
<th>Gray Level(j)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of pixels</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \frac{\sum_{i=1}^{n} k_i}{n} \]

\[ s = \frac{\sum_{i=1}^{n} k_i}{n} \]

\[ s \times 9 \]

### Implementation (3)

<table>
<thead>
<tr>
<th>No. of pixels</th>
<th>Gray level</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

Output image
Gray scale = [0,9]

### Note (1)

- Histogram equalization distributes the graylevels to reach maximum gray (white) because the cumulative distribution function equals 1 when \( 0 \leq r \leq L-1 \)
- If \( \sum_{i=1}^{n} k_i \) is slightly different among consecutive \( k_i \)'s, those graylevels will be mapped to (nearly) identical values as we have to produce an integer grayvalue as output
- Thus, the discrete transformation function cannot guarantee a one-to-one mapping

### Note (2)

- The implementation described above is widely interpreted as histogram equalization.
- It is readily implemented with a LUT.
- It does not produce a strictly flat histogram.
- There is a more accurate solution. However, it may require a one-to-many mapping that cannot be implemented with a LUT.
Better Implementation (1)

```c
void histeq(imageP I1, imageP I2)
{
    int i, R;
    int left[MXGRAY], width[MXGRAY];
    uchar *in, *out;
    long total, Hsum, Havg, histo[MXGRAY];
    /* total number of pixels in image */
    total = (long) I1->width * I1->height;
    /* init I2 dimensions and buffer */
    I2->width  = I1->width;
    I2->height = I1->height;
    I2->image  = (uchar *) malloc(total);
    /* init input and output pointers */
    in   = I1->image; /* input  image buffer */
    out = I2->image; /* output image buffer */
    /* compute histogram */
    for(i=0; i<MXGRAY; i++) histo[i] = 0; /* clear histogram */
    for(i=0; i<total; i++) histo[in[i]]++; /* eval histogram */
    R    = 0; /* right end of interval */
    Hsum = 0; /* cumulative value for interval */
    Havg = total / MXGRAY; /* interval value for uniform histogram */

    /* evaluate remapping of all input gray levels; */
    /* Each input gray value maps to an interval of valid output values. */
    /* The endpoints of the intervals are left[i] and left[i]+width[i]. */
    for(i=0; i<MXGRAY; i++) {
        left[i] = R; /* left end of interval */
        Hsum += histo[i]; /* cum. interval value */
        while(Hsum>Havg && R<MXGRAY-1) { /* make interval wider */
            Hsum -= Havg; /* adjust Hsum */
            R++; /* update right end */
        }
        width[i] = R - left[i] + 1; /* width of interval */
    }
    /* visit all input pixels and remap intensities */
    for(i=0; i<total; i++) {
        if(width[in[i]] == 1) out[i] = left[in[i]];
        else { /* in[i] spills over into width[i] possible values */
            /* randomly pick from 0 to width[i] */
            R = ((rand()&0x7fff)*width[in[i]])>>15; /* 0 <= R < width */
            out[i] = left[in[i]] + R;
        }
    }
}
```

Note

- Histogram equalization has a disadvantage:
  - it can generate only one type of output image.
- With histogram specification we can specify the shape of the histogram that we wish the output image to have.
- It doesn’t have to be a uniform histogram.
- Histogram specification is a trial-and-error process.
- There are no rules for specifying histograms, and one must resort to analysis on a case-by-case basis for any given enhancement task.

Histogram Matching Rule

Let \( v_{out} = T(v_{in}) \) if \( T \) is a unique, monotonic function then

\[
\int_{v_{in}}^{v_{out}} h_{1}(u)du = \int_{v_{in}}^{v_{out}} h_{0}(u)du
\]

This can be restated in terms of the histogram matching rule:

\[
c_{1}(v_{out}) = c_{0}(v_{in})
\]

Where \( c_{1}(v_{out}) = \text{# pixels } \leq v_{out} \) and \( c_{0}(v_{in}) = \text{# pixels } \leq v_{in} \).
This requires that \( v_{out} = c_{1}^{-1}(c_{0}(v_{in})) \)
which is the basic equation for histogram matching techniques.

Histories are Discrete

- Impossible to match all histogram pairs because they are discrete.
Problems with Discrete Case

- The set of input pixel values is a discrete set, and all the pixels of a given value are mapped to the same output value. For example, all six pixels of value one are mapped to the same value so it is impossible to have only four corresponding output pixels.

- No inverse for $c_1$ in $v_{out} = c_1^{-1}(c_0(v_{in}))$ because of discrete domain. Solution: choose $v_{out}$ for which $c_1(v_{out})$ is closest to $c_0(v_{in})$.

- $v_{in} \rightarrow v_{out}$ such that $|c_1(v_{out}) - c_0(v_{in})|$ is a minimum.

Histogram Matching Example (1)

Histogram Matching Example (2)

Implementation (1)

```c
int histogramMatch(imageP I1, imageP histo, imageP I2)
{
    int i, p, R;
    int left[MXGRAY], right[MXGRAY];
    int total, Hsum, Havg, h1[MXGRAY], *h2;
    unsigned char *in, *out;
    double scale;
    /* total number of pixels in image */
    total = (long) I1->height * I1->width;
    /* init I2 dimensions and buffer */
    I2->width = I1->width;
    I2->height = I1->height;
    I2->image = (unsigned char *) malloc(total);
    in = I1->image; /* input image buffer */
    out = I2->image; /* output image buffer */
    for(i=0; i<MXGRAY; i++) h1[i] = 0; /* clear histogram */
    for(i=0; i<total; i++) h1[in[i]]++; /* eval histogram */
    /* target histogram */
    h2 = (int *) histo->image;
    /* normalize h2 to conform with dimensions of I1 */
    for(i=Havg=0; i<MXGRAY; i++) Havg += h2[i];
    scale = (double) total / Havg;
    if(scale != 1) for(i=0; i<MXGRAY; i++) h2[i] *= scale;
    R = 0;
    Hsum = 0;
    /* evaluate remapping of all input gray levels;
    Each input gray value maps to an interval of valid output values.
    The endpoints of the intervals are left[i] and right[i] */
    for(i=0; i<MXGRAY; i++) {
        left[i] = R; /* left end of interval */
        Hsum += h1[i]; /* cumulative value for interval */
        while(Hsum<h2[R] & R<MXGRAY-1) { /* compute width of interval */
            Hsum -= h2[R]; /* adjust Hsum as interval widens */
            R++; /* update */
        }
        right[i] = R; /* init right end of interval */
    }
    /* clear h1 and reuse it below */
    for(i=0; i<MXGRAY; i++) h1[i] = 0;
    /* visit all input pixels */
    for(i=0; i<total; i++) {
        p = left[in[i]]; /* mapping satisfies h2 */
        if(p < left[in[i]]) {
            out[i] = p;
        }
    }
    return 0;
}
```

Implementation (2)

```c
    /* normalize h2 to conform with dimensions of I1 */
    for(i=Havg=0; i<MXGRAY; i++) Havg += h2[i];
    scale = (double) total / Havg;
    if(scale != 1) for(i=0; i<MXGRAY; i++) h2[i] *= scale;
    R = 0;
    Hsum = 0;
    /* evaluate remapping of all input gray levels;
    Each input gray value maps to an interval of valid output values.
    The endpoints of the intervals are left[i] and right[i] */
    for(i=0; i<MXGRAY; i++) {
        left[i] = R; /* left end of interval */
        Hsum += h1[i]; /* cumulative value for interval */
        while(Hsum<h2[R] & R<MXGRAY-1) { /* compute width of interval */
            Hsum -= h2[R]; /* adjust Hsum as interval widens */
            R++; /* update */
        }
        right[i] = R; /* init right end of interval */
    }
```

Implementation (3)
Local Pixel Value Mappings

- Histogram processing methods are global, in the sense that pixels are modified by a transformation function based on the graylevel content of an entire image.
- We sometimes need to enhance details over small areas in an image, which is called a local enhancement.
- Solution: apply transformation functions based on graylevel distribution within pixel neighborhood.

General Procedure

- Define a square or rectangular neighborhood.
- Move the center of this area from pixel to pixel.
- At each location, the histogram of the points in the neighborhood is computed and histogram equalization, histogram matching, or other graylevel mapping is performed.
- Exploit easy histogram update since only one new row or column of neighborhood changes during pixel-to-pixel translation.
- Another approach used to reduce computation is to utilize nonoverlapping regions, but this usually produces an undesirable checkerboard effect.

Example: Local Enhancement

- Original image (slightly blurred to reduce noise)
- Global histogram equalization enhances noise & slightly increases contrast but the structural details are unchanged
- Local histogram equalization using 7x7 neighborhood reveals the small squares inside of the larger ones in the original image.

Definitions (1)

\[
\mu(x, y) = \frac{1}{N} \sum_{i} f(i, j)
\]

mean

\[
\sigma(x, y) = \sqrt{\frac{1}{N} \sum_{i} (f(i, j) - \mu(x, y))^2}
\]

standard deviation

- Let \( p(ri) \) denote the normalized histogram entry for grayvalue \( ri \) for \( 0 \leq i < L \) where \( L \) is the number of graylevels.
- \( \mu \) can be rewritten as

\[
\mu = \frac{1}{N} \sum_{i} ri \cdot p(ri)
\]

Example: Statistical Differencing

- Produces the same contrast throughout the image.
- Stretch \( f(x, y) \) away from or towards the local mean to achieve a balanced local standard deviation throughout the image.
- \( \sigma \) is the desired standard deviation and it controls the amount of stretch.
- The local mean can also be adjusted:

\[
\begin{align*}
\mu_0 & = \mu + \alpha \cdot \mu \\
\sigma_0 & = \beta \cdot \sigma \\
m_0 & = m + \alpha \cdot m
\end{align*}
\]

- Speedups can be achieved by dividing the image into blocks (tiles), exactly computing the mean and standard deviation at the center of each block, and then linearly interpolating between blocks in order to compute an approximation at any arbitrary position. In addition, the mean and standard deviation can be computed incrementally.

Definitions (2)

- The nth moment of \( r \) about its mean is defined as

\[
\mu_n(r) = \sum_{i} (r - m)^n p(r)
\]

- It follows that:

\[
\begin{align*}
\mu_0 & = 1 \\
\mu_1 & = 0 \\
\mu_2 & = \sigma^2
\end{align*}
\]

- The second moment is known as variance \( \sigma^2(r) \)
- The standard deviation is the square root of the variance.
- The mean and standard deviation are measures of average grayvalue and average contrast, respectively.
Example: Local Statistics (1)

The filament in the center is clear. There is another filament on the right side that is darker and hard to see.

Goal: enhance dark areas while leaving the light areas unchanged.

Example: Local Statistics (2)

Solution:
Identify candidate pixels to be dark pixels with low contrast.
Dark: local mean < $k_0 \times$ global mean, where $0 < k_0 < 1$.
Low contrast: $k_1 \times$ global variance < local variance < $k_2 \times$ global variance, where $k_1 < k_2$.
Multiply identified pixels by constant $E>1$. Leave other pixels alone.

Example: Local Statistics (3)

Results for $E=4$, $k_0=0.4$, $k_1=0.02$, $k_2=0.4$. 3x3 neighborhoods used.

Objectives

• In this lecture we describe arithmetic and logic operations commonly used in image processing.
• Arithmetic ops:
  - Addition, subtraction, multiplication, division
  - Hybrid: cross-dissolves
• Logic ops:
  - AND, OR, XOR, BIC, ...

Arithmetic/Logic Operations

Prof. George Wolberg
Dept. of Computer Science
City College or New York

Arithmetic/Logic Operations

• Arithmetic/Logic operations are performed on a pixel-by-pixel basis between two images.
• Logic NOT operation performs only on a single image.
  - It is equivalent to a negative transformation.
• Logic operations treat pixels as binary numbers:
  - $158 \& 235 = 10011110 \& 11101011 = 10001010$
• Use of LUTs requires 16-bit rather than 8-bit indices:
  - Concatenate two 8-bit input pixels to form a 16-bit index into a 64K-entry LUT. Not commonly done.
Addition / Subtraction

Addition:
for(i=0; i<total; i++)
out[i] = MIN(((int)in1[i]+in2[i]), 255);

Subtraction:
for(i=0; i<total; i++)
out[i] = MAX(((int)in1[i]-in2[i]), 0);

Avoid overflow: clip result
Avoid underflow: clip result

Overflow / Underflow

• Default datatype for pixel is unsigned char.
• It is 1 byte that accounts for nonnegative range [0, 255].
• Addition of two such quantities may exceed 255 (overflow).
• This will cause wrap-around effect:
  - 254: 11111110
  - 255: 11111111
  - 256: 100000000
  - 257: 100000001
• Notice that low-order byte reverts to 0, 1, ... when we exceed 255.
• Clipping is performed to prevent wrap-around.
• Same comments apply to underflow (result < 0).

Implementation Issues

• The values of a subtraction operation may lie between -255 and 255. Addition: [0, 510].
• Clipping prevents over/underflow.
• Alternative: scale results in one of two ways:
  1. Add 255 to every pixel and then divide by 2.
     • Values may not cover full [0, 255] range
     • Requires about intermediate image
     • Fast and simple to implement
  2. Add negative of min difference (shift min to 0). Then, multiply all pixels by 255/(max difference) to scale range to [0, 255] interval.
     • Full utilization of [0, 255] range
     • Requires about intermediate image
     • More complex and difficult to implement

Example of Subtraction Operation

Example: Mask Mode Radiography

h(x,y) is the mask, an X-ray image of a region of a patient’s body captured by an intensified TV camera (instead of traditional X-ray film) located opposite an X-ray source
f(x,y) is an X-ray image taken after injection of a contrast medium (iodine) into the bloodstream, with mask subtracted out.

Note:
• the background is dark because it doesn’t change much in both images.
• the difference area is bright because it has a big change

Arithmetic Operations: Cross-Dissolve

• Linearly interpolate between two images.
• Used to perform a fade from one image to another.
• Morphing can improve upon the results shown below.

for(i=0; i<total; i++)
out[i] = in1[i]*f + in2[i]*(1-f);
Masking

- Used for selecting subimages.
- Also referred to as region of interest (ROI) processing.
- In enhancement, masking is used primarily to isolate an area for processing.
- AND and OR operations are used for masking.

Digital Halftoning

Prof. George Wolberg
Dept. of Computer Science
City College of New York

Objectives

- In this lecture we review digital halftoning techniques to convert grayscale images to bitmaps:
  - Unordered (random) dithering
  - Ordered dithering
  - Patterning
  - Error diffusion

Background

- An 8-bit grayscale image allows 256 distinct gray levels.
- Such images can be displayed on a computer monitor if the hardware supports the required number of intensity levels.
- However, some output devices print or display images with much fewer gray levels.
- In these cases, the grayscale images must be converted to binary images, where pixels are only black (0) or white (255).
- Thresholding is a poor choice due to objectionable artifacts.
- Strategy: sprinkle black-and-white dots to simulate gray.
- Exploit spatial integration (averaging) performed by eye.

Thresholding

- The simplest way to convert from grayscale to binary.

Loss of information is unacceptable.
**Unordered Dither (1)**

- Reduce quantization error by adding uniformly distributed white noise (dither signal) to the input image prior to quantization.
- Dither hides objectional artifacts.
- To each pixel of the image, add a random number in the range \([-m, m]\), where \(m\) is MXGRAY/quantization-levels.

**Unordered Dither (2)**

1 bpp 2 bpp 3 bpp 4 bpp

Quantization

Dither/Quantization

**Ordered Dithering**

- Objective: expand the range of available intensities.
- Simulates \(n\) bpp images with \(m\) bpp, where \(n > m\) (usually \(m = 1\)).
- Exploit eye’s spatial integration.
  - Gray is due to average of black/white dot patterns.
  - Each dot is a circle of black ink whose area is proportional to \((1 – \text{intensity})\).
  - Graphics output devices approximate the variable circles of halftone reproductions.

- 2 x 2 pixel area of a bilevel display produces 5 intensity levels.
- \(n \times n\) group of bilevel pixels produces \(n^2 + 1\) intensity levels.
- Tradeoff: spatial vs. intensity resolution.

**Dither Matrix (1)**

- Consider the following 2x2 and 3x3 dither matrices:

  \[
  D^{(2)} = \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix}, \quad D^{(3)} = \begin{bmatrix} 6 & 8 & 4 \\ 1 & 0 & 3 \\ 5 & 2 & 7 \end{bmatrix}
  \]

- To display a pixel of intensity \(I\), we turn on all pixels whose associated dither matrix values are less than \(I\).
- The recurrence relation given below generates larger dither matrices of dimension \(n \times n\), where \(n\) is a power of 2.

  \[
  D^{(n)} = 4D^{(n/2)} + D^{(n/2)} \cdot U^{(n/2)} - \frac{1}{2} \text{round}\left(\frac{D^{(n/2)} \cdot U^{(n/2)}}{2}\right)
  \]

  where \(D^{(1)}\) is an \(n \times n\) matrix of 1’s.

**Dither Matrix (2)**

- Example: a 4x4 dither matrix can be derived from the 2x2 matrix.

**Patterning**

- Let the output image be larger than the input image.
- Quantize the input image to \([0 \ldots n^2]\) gray levels.
- Threshold each pixel against all entries in the dither matrix.
  - Each pixel forms a 4x4 block of black-and-white dots for a \(D^{(n)}\) matrix.
  - An \(n \times n\) input image becomes a 4n x 4n output image.
- Multiple display pixels per input pixel.
- The dither matrix \(D^{(n)}\) is used as a spatially-varying threshold.
- Large input areas of constant value are displayed exactly as before.
Implementation

- Let the input and output images share the same size.
- First quantize the input image to \([0...n^2]\) gray levels.
- Compare the dither matrix with the input image.

```c
for(y=0; y<h; y++) // visit all input rows
  for(x=0; x<w; x++){ // visit all input cols
    i = x % n; // dither matrix index
    j = y % n; // dither matrix index
    // threshold pixel using dither value \(D_{ij}(n)\)
    out[y*w+x] = (in[y*w+x] > D_{ij}(n))? 255 : 0;
  }
```

Error Diffusion

- An error is made every time a grayvalue is assigned to be black or white at the output.
- Spread that error to its neighbors to compensate for over/undershoots in the output assignments
  - If input pixel 130 is mapped to white (255) then its excessive brightness (255-130) must be subtracted from neighbors to enforce a bias towards darker values to compensate for the excessive brightness.
- Like ordered dithering, error diffusion permits the output image to share the same dimension as the input image.

Floyd-Steinberg Algorithm

\[
\begin{align*}
&f(x, y) \quad f^*(x, y) \quad \text{Threshold} \quad g(x, y) \\
&\text{Error} (e(x, y)) \quad o(x, y)
\end{align*}
\]

\[
f'(x, y) = f(x, y) + \sum_{ij} w_{ij}(x-i, y-j) = \text{"corrected intensity value"}
\]

\[
g(x, y) = \begin{cases} 
255 & \text{if } f'(x, y) > \text{MXGRAY}/2 \\
0 & \text{otherwise}
\end{cases}
\]

\[
e(x, y) = f'(x, y) - g(x, y)
\]

\[
\sum_{ij} w_{ij} = 1
\]

Error Diffusion Weights

- Note that visual improvements are possible if left-to-right scanning among rows is replaced by serpentine scanning (zig-zag). That is, scan odd rows from left-to-right, and scan even rows from right-to-left.
- Further improvements can be made by using larger neighborhoods.
- The sum of the weights should equal 1 to avoid emphasizing or suppressing the spread of errors.

\[
\begin{array}{c|c|c|c|c|c|c|c}
\text{Floyd-Steinberg} & 16/11 & 6/11 & 5/11 & 5/11 & 6/11 & 16/11 \\
\text{Jarvis-Judice-Ninke} & 16/7 & 16/7 & 4/1 & 4/1 & 4/1 & 16/7 \\
\text{Stucki} & 16/4 & 16/4 & 4/1 & 4/1 & 4/1 & 16/4 \\
\end{array}
\]

Examples (1)

- Floyd-Steinberg
- Jarvis-Judice-Ninke
- Stucki
Examples (2)

Floyd-Steinberg

Jarvis-Judice-Ninke

Examples (3)

Floyd-Steinberg

Jarvis-Judice-Ninke

Implementation

```
thr = MAXGRAY / 2;  // init threshold value
for(y=0; y<h; y++){  // visit all input rows
  for(x=0; x<w; x++) {  // visit all input cols
    *out = (*in < thr)?  // threshold
      BLACK : WHITE;  // note: use LUT!
    e = *in - *out;  // eval error
    in[ 1 ] +=(e*7/16.);  // add error to E nbr
    in[w-1] +=(e*3/16.);  // add error to SW nbr
    in[w] +=(e*5/16.);  // add error to S  nbr
    in[w+1] +=(e*1/16.);  // add error to SE nbr
    in++;  // advance input  ptr
    out++;  // advance output ptr
  }
}
```

Comments

- Two potential problems complicate implementation:
  - errors can be deposited beyond image border
  - errors may force pixel grayvalues outside the [0,255] range

  True for all neighborhood ops

Solutions to Border Problem (1)

- Perform if statement prior to every error deposit
  - Drawback: inefficient / slow
- Limit excursions of sliding weights to lie no closer than 1 pixel from image boundary (2 pixels for J-J-N weights).
  - Drawback: output will be smaller than input
- Pad image with extra rows and columns so that limited excursions will yield smaller image that conforms with original input dimensions. Padding serves as placeholder.
  - Drawback: excessive memory needs for intermediate image

Solutions to Border Problem (2)

- Use of padding is further undermined by fact that 16-bit precision (short) is needed to accommodate pixel values outside [0, 255] range.
- A better solution is suggested by fact that only two rows are active while processing a single scanline in the Floyd-Steinberg algorithm (3 for J-J-N).
- Therefore, use a 2-row (or 3-row) circular buffer to handle the two (or three) current rows.
- The circular buffer will have the necessary padding and 16-bit precision.
- This significantly reduces memory requirements.
Circular Buffer

New Implementation

thr = MXGRAY /2; // init threshold value

copyRowToCircBuffer(0); // copy row 0 to circular buffer

for(y=0; y<h; y++){ // visit all input rows

copyRowToCircBuffer(y+1); // copy next row to circ buffer

in1 = buf[(y+1)%2] + 1; // circ buffer ptr; skip over pad

in2 = buf[(y+2)%2] + 1; // circ buffer ptr; skip over pad

out = (*in1 < thr)? BLACK : WHITE; // threshold

for(x=0; x<w; x++){ // visit all input cols

*e = *in1 - *out; // eval error

in1[1] += (*e*7/16.); // add error to E nbr

in2[0] += (*e*3/16.); // add error to SW nbr

in2[1] += (*e*5/16.); // add error to S  nbr

in2[2] += (*e*1/16.); // add error to SE nbr

in1++; in2++; // advance circ buffer ptrs

out++; // advance output ptr

}

Objectives

• This lecture describes various neighborhood operations:
  - Blurring
  - Edge detection
  - Image sharpening
  - Convolution

Neighborhood Operations

Prof. George Wolberg
Dept. of Computer Science
City College or New ork

Neighborhood Operations

• Output pixels are a function of several input pixels.
• h(x,y) is defined to weigh the contributions of each input pixel to a particular output pixel.
• g(x,y) = T[f(x,y); h(x,y)]

Spatial Filtering

• h(x,y) is known as a filter kernel, filter mask, or window.
• The values in a filter kernel are coefficients.
• Kernels are usually of odd size: 3x3, 5x5, 7x7
• This permits them to be properly centered on a pixel
  - Consider a horizontal cross-section of the kernel.
  - Size of cross-section is odd since there are 2n+1 coefficients: n neighbors to the left + n neighbors to the right + center pixel

Wolberg: Image Processing Course Notes
Spatial Filtering Process

- Slide filter kernel from pixel to pixel across an image.
- Use raster order: left-to-right from the top to the bottom.
- Let pixels have grayvalues $f_i$.
- The response of the filter at each $(x,y)$ point is:
  $$ R = h_i f_i + h_{i+1} f_{i+1} + \ldots + h_{m+n-1} f_{m+n-1} $$

Linear Filtering

- Let $f(x,y)$ be an image of size $M \times N$.
- Let $h(i,j)$ be a filter kernel of size $m \times n$.
- Linear filtering is given by the expression:
  $$ g(x, y) = \sum_{i=-s}^{s} \sum_{j=-t}^{t} h(i, j) f(x + i, y + j) $$

Spatial Averaging

- Used for blurring and for noise reduction
- Blurring is used in preprocessing steps, such as:
  - removal of small details from an image prior to object extraction
  - bridging of small gaps in lines or curves
- Output is average of neighborhood pixels.
- This reduces the “sharp” transitions in gray levels.
- Sharp transitions include:
  - random noise in the image
  - edges of objects in the image
- Smoothing reduces noise (good) and blurs edges (bad)

3x3 Smoothing Filters

- The constant multiplier in front of each kernel is equal to the sum of the values of its coefficients.
- This is required to compute an average.

Unweighted/Weighted Averaging

- Unweighted averaging (smoothing filter):
  $$ g(x, y) = \frac{1}{m} \sum f(i, j) $$
- Weighted averaging:
  $$ g(x, y) = \sum f(i, j) h(x-i, y-j) $$

Unweighted Averaging

- Unweighted averaging over a 5-pixel neighborhood along a horizontal scanline can be done with the following statement:
  $$ \text{out}[x] = (\text{in}[x-2] + \text{in}[x-1] + \text{in}[x] + \text{in}[x+1] + \text{in}[x+2])/5; $$
- Each output pixel requires 5 pixel accesses, 4 adds, and 1 division. A simpler version (for unweighted averaging only) is:
  $$ \text{sum} = \text{in}[0] + \text{in}[1] + \text{in}[2] + \text{in}[3] + \text{in}[4]; $$
  $$ \text{out}[x] = \text{sum}/5; $$
  $$ \text{sum} = (\text{in}[x] - \text{in}[x-2]); $$
- Limited excursions reduce size of output
Image Averaging

- Consider a noisy image \( g(x,y) \) formed by the addition of noise \( \eta(x,y) \) to an original image \( f(x,y) \):
  \[
g(x,y) = f(x,y) + \eta(x,y)
\]
- If the noise has zero mean and is uncorrelated then we can compute the image formed by averaging \( K \) different noisy images:
  \[
  \bar{g}(x,y) = \frac{1}{K} \sum_{k=1}^{K} g_k(x,y)
  \]
- The variance of the averaged image diminishes:
  \[
  \sigma^2_{\bar{g}(x,y)} = \frac{1}{K} \sigma^2_{g(x,y)}
  \]
- Thus, as \( K \) increases the variability (noise) of the pixel at each location \((x,y)\) decreases assuming that the images are all registered (aligned).

Noise Reduction (1)

- Astronomy is an important application of image averaging.
- Low light levels cause sensor noise to render single images virtually useless for analysis.

Noise Reduction (2)

- Difference images and their histograms yield better appreciation of noise reduction.
- Notice that the mean and standard deviation of the difference images decrease as \( K \) increases.

General Form: Smoothing Mask

- Filter of size \( mn \) (where \( m \) and \( n \) are odd)
  \[
g(x,y) = \frac{1}{\sum_{i=-s}^{s} \sum_{j=-t}^{t} h(i,j)} \sum_{i=-s}^{s} \sum_{j=-t}^{t} h(i,j) f(x+i, y+j)
  \]
- summation of all coefficients of the mask
- Note that \( s = (m-1)/2 \) and \( t = (n-1)/2 \)

Example

- a) original image 500x500 pixel
- b) - f) results of smoothing with square averaging filter of size \( n = 3, 5, 9, 15 \) and 35, respectively.
- Note:
  - big mask is used to eliminate small objects from an image.
  - the size of the mask establishes the relative size of the objects that will be blended with the background.

Example

- Blur to get gross representation of objects.
- Intensity of smaller objects blend with background.
- Larger objects become blob-like and easy to detect.
Unsharp Masking

- Smoothing affects transition regions where grayvalues vary.
- Subtraction isolates these edge regions.
- Adding edges back onto image causes edges to appear more pronounced, giving the effect of image sharpening.

Order-Statistics Filters

- Nonlinear filters whose response is based on ordering (ranking) the pixels contained in the filter support.
- Replace value of the center pixel with value determined by ranking result.
- Order statistic filters applied to \( n \times n \) neighborhoods:
  - median filter: \( R = \text{median} \{ z_k | k = 1,2,\ldots,n^2 \} \)
  - max filter: \( R = \max \{ z_k | k = 1,2,\ldots,n^2 \} \)
  - min filter: \( R = \min \{ z_k | k = 1,2,\ldots,n^2 \} \)

Unsharp Masking Diagram

Median Filter

- Sort all neighborhood pixels in increasing order.
- Replace neighborhood center with the median.
- The window shape does not need to be a square.
- Special shapes can preserve line structures.
- Useful in eliminating intensity spikes: salt & pepper noise.

Median Filter Properties

- Excellent noise reduction
- Forces noisy (distinct) pixels to conform to their neighbors.
- Clusters of pixels that are light or dark with respect to their neighbors, and whose area is less than \( n^2/2 \) (one-half the filter area), are eliminated by an \( n \times n \) median filter.
- \( k \)-nearest neighbor is a variation that blends median filtering with blurring:
  - Set output to average of \( k \) nearest entries around median

Examples (1)

Additive salt & pepper noise  Median filter output  Blurring output

Examples (2)

![Figure 3.17](image.png)
Derivative Operators

- The response of a derivative operator is proportional to the degree of discontinuity of the image at the point at which the operator is applied.
- Image differentiation
  - enhances edges and other discontinuities (noise)
  - deemphasizes area with slowly varying graylevel values.
- Derivatives of a digital function are approximated by differences.

First-Order Derivative

- Must be zero in areas of constant grayvalues.
- Must be nonzero at the onset of a grayvalue step or ramp.
- Must be nonzero along ramps.

\[
\frac{df(x)}{dx} = f(x+1) - f(x)
\]

Second-Order Derivative

- Must be zero in areas of constant grayvalues.
- Must be nonzero at the onset of a grayvalue step or ramp.
- Must be zero along ramps of constant slope.

\[
\frac{d^2 f(x)}{dx^2} = \frac{df(x)}{dx} - \frac{df(x-1)}{dx} = f(x+1) + f(x-1) - 2f(x)
\]

Comparisons

- 1st-order derivatives:
  - produce thicker edges
  - strong response to graylevel steps
- 2nd-order derivatives:
  - strong response to fine detail (thin lines, isolated points)
  - double response at step changes in graylevel

Laplacian Operator

- Simplest isotropic derivative operator
- Response independent of direction of the discontinuities.
- Rotation-invariant: rotating the image and then applying the filter gives the same result as applying the filter to the image first and then rotating the result.
- Since derivatives of any order are linear operations, the Laplacian is a linear operator.

\[
\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}
\]
Discrete Form of Laplacian

\[ \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \]

where

\[ \frac{\partial^2 f}{\partial x^2} = f(x+1,y) + f(x-1,y) - 2f(x,y) \]

\[ \frac{\partial^2 f}{\partial y^2} = f(x,y+1) + f(x,y-1) - 2f(x,y) \]

\[ \nabla^2 f = [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)] \]

Laplacian Mask

<table>
<thead>
<tr>
<th>Isotropic result for rotations in increments of 90°</th>
<th>Isotropic result for rotations in increments of 45°</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 1 1 1</td>
<td>0 1 0 1 1 1</td>
</tr>
<tr>
<td>1 -1 1 1 -1 1</td>
<td>0 -1 0 0 -1 0</td>
</tr>
<tr>
<td>0 -1 0 -1 -1 0</td>
<td>-1 4 -1 8 -1</td>
</tr>
<tr>
<td>-1 0 -1 8 -1</td>
<td>0 -1 0 0 -1 0</td>
</tr>
</tbody>
</table>

Another Derivation

1/9
1 1 1
1 1 1
1 1 1

Unweighted Average Smoothing Filter

Retain Original

1/9
1 -1 -1
1 0 0
1 -1 -1

Original – Average (negative of Laplacian Operator)

In constant areas: 0
Near edges: high values

Effect of Laplacian Operator

- Since the Laplacian is a derivative operator
- It highlights graylevel discontinuities in an image
- It deemphasizes regions with slowly varying gray levels
- The Laplacian tends to produce images that have
  - Grayish edge lines and other discontinuities all superimposed
    on a dark featureless background

Example

Simplification

- Addition of image with Laplacian can be combined into
  one operator:

\[ g(x,y) = f(x,y) - [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) + 4f(x,y)] \]

\[ = 5f(x,y) - [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)] \]

\[ \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \]
Gradient Operator (1)

- The gradient is a vector of directional derivatives.
- Although not strictly correct, the magnitude of the gradient vector is referred to as the gradient.
- First derivatives are implemented using this magnitude:
  \[ \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \]

Approximation:

\[ \nabla f = \frac{1}{\sqrt{2}} \left( \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} + \begin{bmatrix} -\frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \right) \]

Summary (1)

<table>
<thead>
<tr>
<th>Continuous</th>
<th>Digital</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(x)$</td>
<td>$v(i)$</td>
</tr>
<tr>
<td>$\partial f/\partial x$</td>
<td>$\partial v/\partial i$</td>
</tr>
<tr>
<td>$f(x)$ $\approx \nabla^T f(x)$</td>
<td>$v(i) = (v(i-1) - v(i+1))$</td>
</tr>
<tr>
<td>$f(x) = \nabla^T f(x)$</td>
<td>$v(i) = (v(i-1) - v(i+1)) - v(i-2) + v(i+2)$</td>
</tr>
<tr>
<td>$f(x) = \nabla^T f(x)$</td>
<td>$v(i) = (v(i-1) - v(i+1)) - v(i-2) + v(i+2)$</td>
</tr>
<tr>
<td>$f(x) = \nabla^T f(x)$</td>
<td>$v(i) = (v(i-1) - v(i+1)) - v(i-2) + v(i+2)$</td>
</tr>
</tbody>
</table>

The Laplacian is a scalar, giving only the magnitude about the change in pixel values at a point. The gradient gives both magnitude and direction.

Gradient Operator (2)

- The components of the gradient vector are linear operators, but the magnitude is not (square, square root).
- The partial derivatives are not rotation invariant (isotropic), but the magnitude is.
- The Laplacian operator yields a scalar: a single number indicating edge strength at point.
- The gradient is actually a vector from which we can compute edge magnitude and direction.

\[ f_{mag}(i, j) = \sqrt{f_x(i, j)^2 + f_y(i, j)^2} \]

\[ f_{mag}(i, j) = \tan^{-1} \frac{f_y(i, j)}{f_x(i, j)} \text{ where } f_x(i, j) = f(x+1, y) - f(x-1, y) \]

Summary (2)

<table>
<thead>
<tr>
<th>One-dimensional</th>
<th>Two-dimensional</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask = [-1 0 1]</td>
<td>mask = [-1 0 1]</td>
</tr>
<tr>
<td>mask = [0 ]</td>
<td>mask = [0 0 0]</td>
</tr>
<tr>
<td>mask = [1 ]</td>
<td>mask = [1 2 1]</td>
</tr>
</tbody>
</table>

\[ f_{mag}(i, j) = \sqrt{f_x(i, j)^2 + f_y(i, j)^2} \]

\[ f_{mag}(i, j) = \tan^{-1} \frac{f_y(i, j)}{f_x(i, j)} \]

Example (1)

**Example**

**Summary (1)**

<table>
<thead>
<tr>
<th>Continuous</th>
<th>Digital</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(x)$</td>
<td>$v(i)$</td>
</tr>
<tr>
<td>$\partial f/\partial x$</td>
<td>$\partial v/\partial i$</td>
</tr>
<tr>
<td>$f(x)$ $\approx \nabla^T f(x)$</td>
<td>$v(i) = (v(i-1) - v(i+1))$</td>
</tr>
<tr>
<td>$f(x) = \nabla^T f(x)$</td>
<td>$v(i) = (v(i-1) - v(i+1)) - v(i-2) + v(i+2)$</td>
</tr>
<tr>
<td>$f(x) = \nabla^T f(x)$</td>
<td>$v(i) = (v(i-1) - v(i+1)) - v(i-2) + v(i+2)$</td>
</tr>
<tr>
<td>$f(x) = \nabla^T f(x)$</td>
<td>$v(i) = (v(i-1) - v(i+1)) - v(i-2) + v(i+2)$</td>
</tr>
</tbody>
</table>

The Laplacian is a scalar, giving only the magnitude about the change in pixel values at a point. The gradient gives both magnitude and direction.

**Gradient Operator (1)**

- The gradient is a vector of directional derivatives.
- Although not strictly correct, the magnitude of the gradient vector is referred to as the gradient.
- First derivatives are implemented using this magnitude:
  \[ \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \]

Approximation:

\[ \nabla f = \frac{1}{\sqrt{2}} \left( \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} + \begin{bmatrix} -\frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \right) \]

Summary (1)

<table>
<thead>
<tr>
<th>Continuous</th>
<th>Digital</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(x)$</td>
<td>$v(i)$</td>
</tr>
<tr>
<td>$\partial f/\partial x$</td>
<td>$\partial v/\partial i$</td>
</tr>
<tr>
<td>$f(x) = \nabla^T f(x)$</td>
<td>$v(i) = (v(i-1) - v(i+1))$</td>
</tr>
<tr>
<td>$f(x) = \nabla^T f(x)$</td>
<td>$v(i) = (v(i-1) - v(i+1)) - v(i-2) + v(i+2)$</td>
</tr>
<tr>
<td>$f(x) = \nabla^T f(x)$</td>
<td>$v(i) = (v(i-1) - v(i+1)) - v(i-2) + v(i+2)$</td>
</tr>
<tr>
<td>$f(x) = \nabla^T f(x)$</td>
<td>$v(i) = (v(i-1) - v(i+1)) - v(i-2) + v(i+2)$</td>
</tr>
</tbody>
</table>

The Laplacian is a scalar, giving only the magnitude about the change in pixel values at a point. The gradient gives both magnitude and direction.

**Example (1)**

**Summary (2)**

<table>
<thead>
<tr>
<th>One-dimensional</th>
<th>Two-dimensional</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask = [-1 0 1]</td>
<td>mask = [-1 0 1]</td>
</tr>
<tr>
<td>mask = [0 ]</td>
<td>mask = [0 0 0]</td>
</tr>
<tr>
<td>mask = [1 ]</td>
<td>mask = [1 2 1]</td>
</tr>
</tbody>
</table>

\[ f_{mag}(i, j) = \sqrt{f_x(i, j)^2 + f_y(i, j)^2} \]

\[ f_{mag}(i, j) = \tan^{-1} \frac{f_y(i, j)}{f_x(i, j)} \]

Example (1)
Example (2)

- **Goal:** sharpen image and bring out more skeletal detail.
- **Problem:** narrow dynamic range and high noise content makes the image difficult to enhance.
- **Solution:**
  1. Apply Laplacian operator to highlight fine detail
  2. Apply gradient operator to enhance prominent edges
  3. Apply graylevel transformation to increase dynamic range

![Image of whole body bone scan](image)

(Continued)

(c) Sobel image smoothed with a 5 x 5 averaging filter (d) Mask image formed by the product of (c) and (e).

(g) Sharpened image obtained by the sum of (a) and (f). (h) Final result obtained by applying a power-law transformation to (g). Compare (g) and (h) with (a).

(Original image courtesy of G.E. Medical Systems.)

Filtering Theory

Prof. George Wolberg
Dept. of Computer Science
City College or New York

Objectives

- This lecture reviews filtering theory.
  - Linearity and spatial-invariance (LSI)
  - Impulse response
  - Sifting integral
  - Convolution

Definitions

- We use two criteria for filters: linearity and spatial-invariance
  - \[ f(x) \rightarrow \text{Filter} \rightarrow g(x) \]
  - \[ f(x) \rightarrow g(x) \]

Linear:

- \[ af(x) \rightarrow ag(x) \]
- \[ f_1(x) + f_2(x) \rightarrow g_1(x) + g_2(x) \]

or simply,

- \[ af_1(x) + b\delta_2(x) \rightarrow ag_1(x) + b\delta_2(x) \]

Space-invariant (shift-invariant):

- \[ f(x-x) \rightarrow g(x-x) \]
LSI Filter

- Physically realizable filters (lenses) are rarely LSI filters.
  - Optical systems impose a limit in their maximum response.
  - Power can't be negative, imposing a limit on the minimum response.
  - Lens aberrations and finite image area prevent LSI property.
- Nevertheless, close enough to approximate as LSI.

Impulse Response

\[
\delta(x) \quad \text{Filter} \quad h(x)
\]

\[
\delta(x) = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \int_{x-\epsilon}^{x+\epsilon} f(x') \, dx'
\]

\[
f(x) = \int_{-\infty}^{\infty} f(\lambda) \delta(x - \lambda) \, d\lambda
\]

\[
h(x) = \sum_{n=-\infty}^{\infty} f(n) \delta(x - n)
\]

Convolution

- Any input signal can be represented by an infinite sum of shifted and scaled impulses:

\[
g(x) = \int_{-\infty}^{\infty} f(x') h(x-x') \, dx'
\]

Discrete Case

- Input function is a scaled shifted version of impulse response:
  \[ f \ast h(x) \]
  \[ f \circ h(x) \]
- \( \ast \): convolution operator
  \( \circ \): filter kernel
- \( h(x) \): convolution kernel
- If \( h(x) = \delta(x) \) then we have an ideal filter: output = input.
- Usually \( h(x) \) extends over several neighbors.
- Discrete convolution:

\[
g(x) = \sum_{n=-\infty}^{\infty} f(n) h(x-n)
\]

Example: Triangle Filter

- Linearity rule: scale \( h(x) \) according to \( f(x) \) and add \( g_1 \), \( g_2 \).
- Obtain \( f(x) \) for any \( x \) by sampling the reconstructed \( g(x) \).

Convolution Summation

- \( g(x) \) is a continuous convolution summation to compute at particular values at \( x \):

\[
g(x) = \sum_{n=-\infty}^{\infty} f(n) h(x-n)
\]

\[
g(x) = f_1 h(x-x_1) + f_2 h(x-x_2)
\]

If \( x_1 = 0 \) and \( x_2 = 1 \) then \( g(x) = f_1(1-x) + f_2 x \)
A Closer Look At the Convolution Integral

Mirrored Kernel

• Why fold over (mirror) kernel \( h(x) \)?
• Why not use \( g(x) = \sum f(\lambda)h(\lambda-x) \)?
By construction: \( f[h(x-x_0)] + f[h(x-x_1)] \)
\( \sum f(\lambda)h(\lambda-x) \)
By centering \( h \) at \( x \): \( f[h(x-x_0)] + f[h(x-x_1)] \)
(which is wrong) Therefore flip \( h \) before centering at \( x \).

• We typically use symmetric kernels: \( h(-x) = h(x) \)

Impulse Function

• Impulse function (or Dirac delta function) is defined as
\[
\delta(x) = \lim_{\varepsilon \to 0} f(x)dx = 1, \quad x = 0
\]
\[
0, \quad x \neq 0
\]
• It can be used to sample a continuous function \( f(x) \) at any point \( x_0 \) as follows:
\[
f(x_0) = \int f(\lambda)\delta(x_0-\lambda)d\lambda = f(x_0)\delta(0) = f(x_0)
\]
\[
\delta(x_0-\lambda) = 0 \text{ for } \lambda \neq x_0
\]

Impulse Response

• When an impulse is applied to a filter, an altered impulse, (the impulse response) is generated at the output.

Sifting Integral

• Any continuous input signal can be represented in the limit by an infinite sum of shifted and scaled impulses.
• This is an outcome of the sifting integral:
\[
f(x) = \int f(\lambda)\delta(x-\lambda)d\lambda
\]
which uses signal \( f(x) \) to scale the collection of impulses:

Convolution Integral (1)

• The response \( g(x) \) of a digital filter to an arbitrary input signal \( f(x) \) is expressed in terms of the impulse response \( h(x) \) of the filter by means of convolution integral:
\[
g(x) = f(x) * h(x) = \int f(\lambda)h(x-\lambda)d\lambda
\]
- where \( * \) denotes the convolution operation,
- \( h(x) \) is used as the convolution (filter) kernel, and
- \( \lambda \) is the dummy variable of integration.
• Kernel \( h(x) \) is treated as a sliding window that is shifted across the entire input signal.
• As \( h(x) \) makes its way across \( f(x) \), a sum of the pointwise products between the two functions is taken and assigned to output \( g(x) \).
Convolution Integral (2)

- This process, known as convolution, is of fundamental importance to linear filtering theory.
- It simply states that the output of an LSI filter will be a superposition of shifted and scaled impulse responses.
- This is used to explain how a continuous image is blurred by a camera as it passes through the lens.
  - In this context, \( h(x) \) is known as the point spread function (PSF), reflecting the limitation of the camera to accurately resolve a small point without somewhat blurring it.

Evaluation

- We can arrive at \( g(x) \) in two ways:
  - Graphical construction of a series of shifted/scaled impulse responses
  - Computing the convolution summation at all points of interest
- Graphical construction more closely follows the physical process as an input signal passes through a filter.
- Convolution summation more closely follows the practical evaluation of \( g(x) \) at a finite number of desired points.
  - Instead of adding scaled and shifted unit impulse responses (responses of unit impulses), we center the unit impulse response at the point of interest.

Graphical Construction

- We can arrive at \( g(x_0) \) in two ways:
  - Graphical construction of a series of shifted/scaled impulse responses
  - Computing the convolution summation at all points of interest
- Graphical construction more closely follows the physical process as an input signal passes through a filter.
- Convolution summation more closely follows the practical evaluation of \( g(x) \) at a finite number of desired points.
  - Instead of adding scaled and shifted unit impulse responses (responses of unit impulses), we center the unit impulse response at the point of interest.

Convolution Summation (1)

- Instead of adding scaled/shifted unit impulse responses, center unit impulse response at point of interest:
  - As \( \lambda \) increases, we scan \( f_j \) from left to right. However, \( h(x_0 - \lambda) \) is scanned from right to left.
  - Reason: points to the left of \( x_0 \) had contributed to \( x_0 \) through the right side of \( h \) (see graphical construction).

Convolution Summation (2)

- More straightforward: implement convolution if both the input and the kernel were scanned in the same direction.
- This permits direct pointwise multiplication among them.
- Thus, flip kernel before centering it on output position.

Convolution Summation (3)

- \( \sum \lambda h_j \) will yield the value for \( g(x_j) \), where \( h_j = h(x_j-x_j) \).
- Rationale: multiplying the unit impulse response \( h \) with \( f_i \) \( h \) is being scaled. The distance between \( x_0 \) and \( x_i \) accounts for the effect of a shifted response function on the current output.
- For most impulse response functions, \( h \) will taper off with increasing distance from its center.
- If the kernel is symmetric (\( h(x) = h(-x) \)), it is not necessary to flip the kernel before centering.
The kernel is often a discrete set of weights, i.e., a 3x3 filter kernel. As long as the kernel is shifted in pixel (integer) increments across the image, there is no alignment problem with underlying image. However, for noninteger pixel increments the kernel values may have no corresponding image values.

\[
g(x) = \sum f(\lambda)h(x-\lambda) \quad \text{for integer values of } \lambda \text{ and arbitrary values of } x
\]

- The kernel is aligned with \( f \) (it is centered at pixel \( f_3 \)).
- The kernel is not aligned with \( f \) (it has been shifted by a ½-pixel increment).

Discrete Convolution (2)

There are two possible solutions to this problem:

1. Represent \( h \) in analytic form for evaluation anywhere. For example, let bell-shape PSF be \( h(y)=2-4y^2 \). The appropriate weight to apply to \( f_i \) can be obtained by setting \( x \) to the difference between the center of the bell (impulse response) and the position of the \( f_i \).

2. Supersample \( h \) so that a dense set of samples are used to represent \( h \). The supersampled version will then be aligned with the image data, or at least it will have values that are nearby.

Objectives

- This lecture reviews Fourier transforms and processing in the frequency domain.
  - Definitions
  - Fourier series
  - Fourier transform
  - Fourier analysis and synthesis
  - Discrete Fourier transform (DFT)
  - Fast Fourier transform (FFT)

Background (1)

- Fourier proved that any periodic function can be expressed as the sum of sinusoids of different frequencies, each multiplied by a different coefficient. \( \rightarrow \) Fourier series
- Even aperiodic functions (whose area under the curve is finite) can be expressed as the integral of sinusoids multiplied by a weighting function. \( \rightarrow \) Fourier transform
- In a great leap of imagination, Fourier outlined these results in a memoir in 1807 and published them in *La Theorie Analytique de la Chaleur* (The Analytic theory of Heat) in 1822. The book was translated into English in 1878.
Example

Useful Analogy

• A glass prism is a physical device that separates light into various color components, each depending on its wavelength (or frequency) content.
• The Fourier transform is a mathematical prism that separates a function into its frequency components.

Fourier Series for Periodic Functions

Fourier Transform for Aperiodic Functions

Fourier Analysis and Synthesis

• Fourier analysis: determine amplitude & phase shifts
• Fourier synthesis: add scaled and shifted sinusoids together
• Fourier transform pair:
  - Forward F.T. $F(u) = \int f(x)e^{-2\pi iux}dx$
  - Inverse F.T. $f(x) = \int F(u)e^{2\pi iux}dx$
where $i = \sqrt{-1}$, and $e^{2\pi iux} = \cos(2\pi ux) \pm j\sin(2\pi ux)$ ← complex exponential at freq. $u$

Fourier Coefficients

• Fourier coefficients $F(u)$ specify, for each frequency $u$, the amplitude and phase of each complex exponential.
• $F(u)$ is the frequency spectrum.
• $f(x)$ and $F(u)$ are two equivalent representations of the same signal.
  - $F(u) = R(u) + iI(u)$
  - $|F(u)| = \sqrt{R^2(u) + I^2(u)}$ ← magnitude spectrum; aka Fourier spectrum
  - $\Phi(u) = \tan^{-1} \frac{I(u)}{R(u)}$ ← phase spectrum
  - $P(u) = |F(u)|^2 = R^2(u) + I^2(u)$ ← spectral density
**1D Example**

For periodic signals, we have the Fourier series:

\[ f(x) = \sum_{n} c(n) e^{j2\pi nx/L} \]

where \( c(n) \) is the \( n \)th Fourier coefficient.

That is, the periodic signal contains all the frequencies that are harmonics of the fundamental frequency.

**2D Example**

**Fourier Series (1)**

For periodic signals, we have the Fourier series:

\[ f(x) = \sum_{n} c(n) e^{j2\pi nx/L} \]

where \( c(n) \) is the \( n \)th Fourier coefficient.

That is, the periodic signal contains all the frequencies that are harmonics of the fundamental frequency.

**Fourier Series (2)**

For periodic signals, we have the Fourier series:

\[ f(x) = \sum_{n} c(n) e^{j2\pi nx/L} \]

where \( c(n) \) is the \( n \)th Fourier coefficient.

That is, the periodic signal contains all the frequencies that are harmonics of the fundamental frequency.

**Fourier Series (3)**

- The Fourier transform is applied for aperiodic signals.
- It is represented as an integral over a continuum of frequencies.
- The Fourier Series is applied for periodic signals.
- It is represented as a summation of frequency components that are integer multiples of some fundamental frequency.

**Example**

Ex: Rectangular Pulse Train

\[ f(x) = \begin{cases} A & \text{in interval} [-W/2, W/2] \\ 0 & \text{otherwise} \end{cases} \]
Discrete Fourier Transform

\[ F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x)e^{-2\pi i xu/N} \]  
forward DFT

\[ f(x) = \sum_{u=0}^{N-1} F(u)e^{2\pi i xu/N} \]  
inverse DFT

for \(0 \leq u \leq N-1\) and \(0 \leq x \leq N-1\) where \(N\) is the number of equi-spaced input samples. The \(1/N\) factor can be in front of \(f(x)\) instead.

Fourier Analysis Code

- DFT maps \(N\) input samples of \(f\) into the \(N\) frequency terms in \(F\).

```c
for(u=0; u<N; u++) { /*compute spectrum over all freq. u */
real = imag = 0; /*reset real, imag component of F(u)*/
for(x=0; x<N; x++) { /* visit each input pixel */
real += (f[x]*cos(-2*PI*u*x/N));
imag += (f[x]*sin(-2*PI*u*x/N));
/* Note: if f is complex, then */
real += (fr[x]*cos()) - (fi[x]*sin());
imag += (fr[x]*sin()) + (fi[x]*cos());

/*because (fr+ifi)(gr+igi)=(frgr-figi)+i(figr+frgi)*/
}
Fr[u] = real / N;
Fi[u] = imag / N;
}
```

Fourier Synthesis Code

```c
for(x=0; x<N; x++) {/* compute each output pixel */
real = imag = 0; /* reset real, imaginary component */
for(u=0; u<N; u++) {
c = cos(2*PI*u*x/N);
s = sin(2*PI*u*x/N);
real += (Fr[u]*c+Fi[u]*s);
imag += (Fr[u]*s+Fi[u]*c);
}
fr[x] = real; /* OR f[x] = sqrt(real*real + imag*imag);
fi[x] = imag;*/
```

Example: Fourier Analysis (1)

- Gibbs phenomenon

Example: Fourier Analysis (2)

- Summary

\[ \mathcal{F}_T \]  
Continuous
\[ \mathcal{F}_D \]  
Discrete
\[ \mathcal{F}_S \]  
Periodic
\[ \mathcal{F}_A \]  
Aperiodic
\[ \mathcal{F}_C \]  
Continuous
\[ \mathcal{F}_D \]  
Discrete
\[ \mathcal{F}_S \]  
Periodic
\[ \mathcal{F}_A \]  
Aperiodic
\[ \mathcal{F}_C \]  
Continuous
\[ \mathcal{F}_D \]  
Discrete
\[ \mathcal{F}_S \]  
Periodic
\[ \mathcal{F}_A \]  
Aperiodic
\[ \mathcal{F}_C \]  
Continuous

Note: \(n=\)kT, \(s[n]=s(nT),\) N = # of samples
2D Fourier Transform

Continuous:
\[ F(f(x,y)) = F(u,v) = \int \int f(x,y)e^{-j2\pi(ux+vy)} \, dx \, dy = \int \int f(x,y)e^{-j2\pi(ux+vy)} \, dx \, dy \]

\[ F^{-1}(F(u,v)) = f(x,y) = \int \int F(u,v)e^{j2\pi(ux+vy)} \, du \, dv = \int \int F(u,v)e^{j2\pi(ux+vy)} \, du \, dv \]

Separable: \( F(u,v) = F(u)F(v) \)

Discrete:
\[ F(u,v) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n)e^{-j2\pi(mu+nv)/MN} \]

\[ f(x,y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} F(u,v)e^{j2\pi(mx+ny)/MN} \]

Separable Implementation

\[ F(u,v) = \frac{1}{N} \sum_{n=0}^{N-1} f(n,v)e^{-j2\pi nx/N} \]

\[ = \frac{1}{M} \sum_{m=0}^{M-1} F(u,m)e^{-j2\pi mu/M} \]

The 2D Fourier transform is computed in two passes:
1) Compute the transform along each row independently.
2) Compute the transform along each column of this intermediate result.

Properties

• Edge orientations in image appear in spectrum, rotated by 90°.
• 3 orientations are prominent: 45°, -45°, and nearly horizontal long white element.

Magnitude and Phase Spectrum

Role of Magnitude vs Phase (1)

Pictures reconstructed using the Fourier phase of another picture

Role of Magnitude vs Phase (2)
Noise Removal

Fast Fourier Transform (1)

- The DFT was defined as:
  \[ F_n(x) = \frac{1}{N} \sum_{k=0}^{N-1} f(k) e^{-\frac{2\pi i kn}{N}} \quad 0 \leq n \leq N - 1 \]
  
  Rewriting:
  \[ F_n = \frac{1}{N} \sum_{k=0}^{N-1} f(k) e^{-\frac{2\pi i kn}{N}} \quad 0 \leq n \leq N - 1 \]
  
  Let \( F_n = \sum_{k=0}^{N-1} f(k) W^{kn} \)
  
  where \( W = e^{-\frac{2\pi i}{N}} = \cos \left( \frac{2\pi}{N} \right) + i \sin \left( \frac{2\pi}{N} \right) \)
  
  Also, let \( N' = 2^r \) (\( N \) is a power of 2)

Fast Fourier Transform (2)

- \( W^N \) can be thought of as a 2D array, indexed by \( n \) and \( k \).
- It represents \( N \) equispaced values along a sinusoid at each of \( N \) frequencies.
- For each frequency \( n \), there are \( N \) multiplications (\( N \) samples in sine wave of freq. \( n \)). Since there are \( N \) frequencies, DFT: \( O(N^2) \)
- With the FFT, we will derive an \( O(N \log N) \) process.

Computational Advantage

Danielson–Lanczos Lemma (1)

1942:
\[ F_e = \sum f_k e^{-\frac{2\pi i kn}{N}} \]
\[ F_o = \sum f_k e^{-\frac{2\pi i (k+n)N}{2N}} \]

Even Numbered Terms \quad Odd Numbered Terms
\[ f_0, f_2, f_4, \ldots \quad f_1, f_3, f_5, \ldots \]

Danielson–Lanczos Lemma (2)

- \( F_n \) can be thought of as a 2D array, indexed by \( n \) and \( k \).
- It represents \( N \) equispaced values along a sinusoid at each of \( N \) frequencies.
- For each frequency \( n \), there are \( N \) multiplications (\( N \) samples in sine wave of freq. \( n \)). Since there are \( N \) frequencies, DFT: \( O(N^2) \)
- With the FFT, we will derive an \( O(N \log N) \) process.

Divide-and-Conquer solution: Solving a problem (\( F_n \)) is reduced to 2 smaller ones.
Potential Problem: \( n \) in \( F_e \) and \( F_o \) is still made to vary from 0 to \( N-1 \). Since each sub-problem is no smaller than original, it appears wasteful.
Solution: Exploit symmetries to reduce computational complexity.
Danielson–Lanczos Lemma (3)

Given: a DFT of length $N$, $F_{x,n} = F_n$

Proof: $F_{x,n} = \sum_{k=0}^{N-1} f_k e^{-i2\pi nk/N} = \sum_{k=0}^{N-1} f_k e^{-i\pi nk/2}$

$W_{n,k} = \cos\left(-\frac{2\pi}{N} \left( k + \frac{N}{2} \right) \right) + i\sin\left(-\frac{2\pi}{N} \left( k + \frac{N}{2} \right) \right)$

$W_{n,k} = \cos\left(-\frac{2\pi}{N} \left( k - \frac{N}{2} \right) \right) - i\sin\left(-\frac{2\pi}{N} \left( k - \frac{N}{2} \right) \right)$

$= -W_{n,k}^*$

Main Points of FFT

$F_n = \sum_{k=0}^{N-1} f_k e^{-i2\pi nk/N}$

$F_n = F_n^* + W_nF_n^*$

$F_n = F_n^* - W_nF_n^*$

But $0 \leq n < N$

$F_n^*$

FFT Example (1)

Input: 10, 15, 20, 25, 5, 30, 8, 4

Weights

DFT is a convolution with kernel values $e^{-i2\pi nk/N}$

These values are derived from a unit circle.

FFT Example (2)

Input: 10, 5, 20, 8, 15, 30, 25, 4

DFT Example (1)

Input: 10, 15, 20, 25, 5, 30, 8, 4
DFT Example (2)

\[
F_1 = 10e^{-i2\pi/5} + 15e^{-i2\pi/5} + 20e^{-i2\pi/5} + \ldots + 5e^{-i2\pi/5} + 4e^{-i2\pi/5} = 10.4 + 180.4 + 25.4 + 30.4 + 8.4 + 4 = -31
\]

\[
F_2 = 10e^{-i2\pi/5} + 15e^{-i2\pi/5} + 20e^{-i2\pi/5} + \ldots + 5e^{-i2\pi/5} + 4e^{-i2\pi/5} = 10.4 + 180.4 + 25.4 + 30.4 + 8.4 + 4 = 31
\]

\[
F_3 = 10e^{-i2\pi/5} + 15e^{-i2\pi/5} + 20e^{-i2\pi/5} + \ldots + 5e^{-i2\pi/5} + 4e^{-i2\pi/5} = 10.4 + 180.4 + 25.4 + 30.4 + 8.4 + 4 = 31
\]

\[
F_4 = 10e^{-i2\pi/5} + 15e^{-i2\pi/5} + 20e^{-i2\pi/5} + \ldots + 5e^{-i2\pi/5} + 4e^{-i2\pi/5} = 10.4 + 180.4 + 25.4 + 30.4 + 8.4 + 4 = 31
\]

\[
F_5 = 10e^{-i2\pi/5} + 15e^{-i2\pi/5} + 20e^{-i2\pi/5} + \ldots + 5e^{-i2\pi/5} + 4e^{-i2\pi/5} = 10.4 + 180.4 + 25.4 + 30.4 + 8.4 + 4 = 31
\]

Filtering in the Frequency Domain

Prof. George Wolberg
Dept. of Computer Science
City College or New York
**Objectives**

- This lecture reviews frequency domain filtering.
  - Convolution theorem
  - Frequency bands
  - Lowpass filter
    - Ideal, Butterworth, Gaussian
  - Highpass filter
    - Ideal, Butterworth, Gaussian
  - Notch filter
  - Homomorphic filtering

**Basics**

1. Multiply input image by \((-1)^{x+y}\) to center the transform to \(u = M/2\) and \(v = N/2\) (if \(M\) and \(N\) are even numbers, then shifted coordinates will be integers)
2. Compute \(F(u,v)\), the DFT of the image from (1)
3. Multiply \(F(u,v)\) by a filter function \(H(u,v)\)
4. Compute the inverse DFT of the result in (3)
5. Obtain the real part of the result in (4)
6. Multiply the result in (5) by \((-1)^{x+y}\) to cancel the multiplication of the input image.

**Convolution Theorem**

"Multiply \(F(u,v)\) by a filter function \(H(u,v)\)"

This step exploits the convolution theorem:

\[
\text{Convolution in one domain is equivalent to multiplication in the other domain.}
\]

\[
f(x) * g(x) \leftrightarrow F(u) \cdot G(u)
\]

\[
f(x) \cdot g(x) \leftrightarrow F(u) * G(u)
\]

**Periodicity**

\[
F(u,v) = F(u+M, v) = F(u,v+N) = F(u+M,v+N)
\]

**Comments**

- The magnitudes from \((M/2)+1\) to \(M-1\) are reflections of the values in the half period to the left of the origin.
- DFT is formulated for values of \(u\) in the interval \([0, M-1]\).
- This yields two back-to-back half periods in this interval.
- For one full period, move origin to \(u=M/2\) by multiplying \(f(x)\) by \((-1)^u\).
Significance of Periodicity

$x$ ranges from 0 to 799 for $x/y$ to slide past $f$

Period too short

Padding (1)

- Corrupted results were due to inadequate period.
- Solution: add padding to increase period.

Padding (2)

Example

Frequency Bands

- Low frequencies: general graylevel appearance over smooth areas: slowly varying grayscales.
- High frequencies: responsible for detail, such as edges and noise.
- A filter that attenuates high frequencies while “passing” low frequencies is a lowpass filter.
- A filter that attenuates low frequencies while “passing” high frequencies is a highpass filter.
- Lowpass filters blur images.
- Highpass filters highlight edges.

Lowpass and Highpass Filters
Improved Highpass Output

- Add constant to filter so that it will not eliminate $F(0,0)$

**Notch Filter**

- $F(0,0)$ represents the average value.
- If $F(0,0) = 0$, then average will be 0.
- Achieved by applying a notch filter:
  
  $$H(u, v) = \begin{cases} 
  0 & \text{if } (u, v) = (M/2, N/2) \\
  1 & \text{otherwise}
  \end{cases}$$
- In reality the average of the displayed image can’t be zero as it needs to have negative gray levels. The output image needs to scale the graylevel.
- Filter has notch (hole) at origin.

**Ideal Lowpass Filter**

- Most sharp detail in this image is contained in the 8% power removed by filter.
- Ringing behavior is a characteristic of ideal filters.
- Little edge info contained in upper 0.5% of spectrum power in this case.

**Image Power Circles**

- Most sharp detail in this image is contained in the 8% power removed by filter.
- Ringing behavior is a characteristic of ideal filters.
- Little edge info contained in upper 0.5% of spectrum power in this case.
The document discusses various types of lowpass filters in image processing, focusing on their characteristics and properties. Here are the key points:

### ILPF Ringing
- Ringing in spatial filter has two characteristics:
  - Dominant component at origin and concentric circular components.
  - Center component responsible for blurring.
  - Concentric components responsible for ringing.
  - Radius of center component and # circles/unit distance are inversely proportional to cutoff frequency.

### Butterworth Lowpass Filter (BLPF)
- The Butterworth lowpass filter transfer function is:
  \[ H(u, v) = \frac{1}{1 + [D(u, v)/D_0]^n} \]
  where \( n \) is the filter order.

### Varying BLPF Cutoff Frequencies
- Unlike the ILPF, BLPF does not have a sharp discontinuity at the cutoff frequency.
- At \( D_0 \), \( H(u, v) \) is down 50% from max value of 1.
- Smooth transition in blurring as cutoff frequency increases.
- No ringing due to filter’s smooth transition between low and high frequencies.

### Spatial Representation of BLPFs
- Varying BLPF cutoff frequencies can be visualized with different levels of ringing.
- No ringing, Imperceptible ringing, Significant ringing.

### Gaussian Lowpass Filter (GLPF)
- The Gaussian lowpass filter transfer function is:
  \[ H(u, v) = e^{-D(u, v)/2\sigma^2} \]
  where \( \sigma = D_0 \) is the cutoff frequency.

### Varying GLPF Cutoff Frequencies
- Smooth transition in blurring as cutoff frequency increases.
- GLPF did not achieve as much smoothing as BLPF of order 2 for same cutoff freq.
- GLPF profile is not as tight as profile of the BLPF of order 2.
- No ringing.
- BLPF is more suitable choice if tight control is needed around cutoff frequency. Only drawback: possible ringing.
Example (1)

Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a data using "90" as 1990 rather than the year 2000.

Example (2)

Example (3)

• Crude but simple way of removing prominent scan lines.
• Blurs out detail while leaving large features recognizable.
• Averages out features smaller than the ones of interest.

Highpass Filters: 1-H<sub>LP</sub>

Ideal highpass filter

\[ H(u,v) = \begin{cases} 0 & \text{if } D(u,v) \leq D_0 \\ 1 & \text{if } D(u,v) > D_0 \end{cases} \]

Butterworth highpass filter

\[ H(u,v) = \frac{1}{1 + \left(D(u,v)/D_0\right)^n} \]

Gaussian highpass filter

\[ H(u,v) = 1 - e^{-D^2(u,v)/2\sigma^2} \]

Spatial Representation of Filters

Ideal | Butterworth | Gaussian

Varying IHPF Cutoff Frequencies

Figure 4.24 Results of ideal highpass filtering the image in Fig. 4.11(a) with \( D_0 = 30 \), 30, and 80, respectively. Problems with ringing are quite evident at 30 and 80.
Varying BHPF Cutoff Frequencies

Example

Varying GHPF Cutoff Frequencies

Homomorphic Filtering (1)

Example

Homomorphic Filtering (2)
Correlation

- Identical to convolution except that kernel is not flipped.
- Kernel is now referred to as the template.
- It is the pattern that we seek to find in the larger image.

\[ f(x, y) \ast h(x, y) \iff F^*(u, v)H(u, v) \]

Example (1)

Example (2)

Reference image

Template

Correlation image

Peak corresponds to best match location

Sampling Theory

Prof. George Wolberg
Dept. of Computer Science
City College or New York

Objectives

- In this lecture we describe sampling theory:
  - Motivation
  - Analysis in the spatial and frequency domains
  - Ideal and nonideal solutions
  - Aliasing and antialiasing
  - Example: oversampling CD players

Sampling Theory

Sampling theory addresses the two following problems:
1. Are the samples of \( g_s(x) \) sufficient to exactly describe \( g(x) \)?
2. If so, how can \( g(x) \) be reconstructed from \( g_s(x) \)?
**Motivation**

- Improper consideration leads to jagged edges in magnification and Moire effects in minification.

**Insight: Analysis in Frequency Domain**

\[
\int_{-\infty}^{\infty} f(x) e^{j2\pi f x \, dt} = \int_{-\infty}^{\infty} f(x) e^{-j2\pi f x \, dt}
\]

**Exploit Well-Known Properties of Fourier Transforms**

1. Multiplication in spatial domain ↔ convolution in frequency domain.
2. Fourier transform of an impulse train is itself an impulse train.
3. \( G_s(f) \) is the original spectrum \( G(f) \) replicated in the frequency domain with period \( f_s \).

\[
G_s(f) = G(f) + G_{high}(f) \rightarrow \text{introduced by sampling}
\]

**Solution to Reconstruction**

Discard replicas \( G_{high}(f) \), leaving baseband \( G(f) \) intact.

Two Provisions for exact reconstruction:

- The signal must be bandlimited. Otherwise, the replicas would overlap with the baseband spectrum.
- \( f_s > 2f_{\text{max}} \) (Nyquist frequency)

**Ideal Filter in the Spatial Domain: Sinc Function**

- The ideal lowpass filter in the spatial domain is the inverse Fourier transform of the ideal box:
**Reciprocal Relationship**

• Reciprocal relationship between spatial and frequency domains:

• For interpolation, \( A \) must equal 1 (unity gain when sinc is centered at samples; pass through 0 at all other samples)

\[ W = \frac{f_{\text{max}}}{2} \text{ cycle / pixel for all digital images} \]

Highest freq: on-off sequence
1 cycle is 2 pixels (black, white, black, white), therefore \( \frac{1}{2} \) cycle per pixel is \( f_{\text{max}} \).

\[
\begin{array}{cccc}
\text{A1} & \frac{1}{2w1} \\
\text{A2} & \frac{1}{2w2} \\
\text{A2} & < \text{A1} \\
\text{W2} & < \text{W1}
\end{array}
\]

\( \text{Note: sinc requires infinite number of neighbors impossible} \)

**Blurring**

• To blur an image, \( W \downarrow \) and \( A \downarrow \) so that \( A/2W \) remains at 1.

• To interpolate among sparser samples, \( A=1 \) and \( W \downarrow \) which means that \( (A/2W) \uparrow \). Since sampling decreases the amplitude of the spectrum, \( (A/2W) \uparrow \) serves to restore it.

\[
\phi(x) = \text{sinc}(x) \Rightarrow \phi(2x) \Rightarrow \phi(x/2)
\]

**Nonideal Reconstruction**

• Possible solution to reconstruction in the spatial domain: truncated sinc.

[Diagram of truncated sinc with ripples and Gibbs phenomenon]

• Alternate solution: nonideal reconstruction

[Diagram showing passband and nonideal reconstruction]

• Nonideal because it attenuates the higher frequencies in the passband and doesn’t fully suppress the stopband.

**Aliasing**

If \( f_s < 2f_{\text{max}} \) our signal is undersampled.

If \( f_s \) is infinite, our signal is not bandlimited.

Either way, we have aliasing: high frequencies masquerade, or alias as, low frequencies

**Antialiasing**

Two approaches to antialiasing for combating aliasing:

1. Sample input at higher rate: increase \( f_s \)

2. Bandlimit input before sampling at low \( f_s \)

**Intuition for increasing \( f_s \):**

Higher sampling rates permit sloppier reconstruction filters.
Nonideal Reconstruction Filter

\[ H_r(f) = \begin{cases} 1 & 0 \leq |f| \leq f_{max} \\ 0 & |f| > f_s \\ \frac{f_{max}}{|f|} & f_{max} > |f| > f_s \\ \frac{f_{max}}{|f|} & f_{max} < |f| \leq f_s \\ \end{cases} \]

Example: Oversampling CD players

\[ f_s = 44.1 \text{ kHz (sample/second)} \]
\[ f_{max} = 20 \text{ kHz} \times 2 + 4.1 \text{ kHz} = 44.1 \text{kHz} \]

Adequate Sampling

Inadequate Sampling

Antialiasing

Image Resampling
Objectives

• In this lecture we review image resampling:
  - Ideal resampling
  - Mathematical formulation
  - Resampling filter
  - Tradeoffs between accuracy and complexity
  - Software implementation

Definition

• Image Resampling: Process of transformation a sampled image from one coordinate system to another.

Mathematical Formulation (1)

Stages:

<table>
<thead>
<tr>
<th>Math Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(x,u)$, $u \in z$</td>
<td>Discrete Input</td>
</tr>
<tr>
<td>$f^\prime = f(x) * r(u) = \sum f(u) r(u - k)$</td>
<td>Reconstruction Input</td>
</tr>
<tr>
<td>$g^\prime = g(x) = r^\prime(m(x))$</td>
<td>Warped Signal</td>
</tr>
<tr>
<td>$g (x) = \sum g (x) * h(u) = \int g (u) h (x - u) du$</td>
<td>Continuous Output</td>
</tr>
<tr>
<td>$g (x) = g^\prime(x) \delta(x)$</td>
<td>Discrete Output</td>
</tr>
</tbody>
</table>

Two filtering components: reconstruction and prefiltering (bandlimiting warped signal before sampling). Cascade them into one by working backwards from $g(x)$ to $f(x)$:

$g(x) = g^\prime(x)$ for $x \in z$

$g(x) = \int f^\prime (m(x)) h(x - u) du = \int \sum f(u) r(m(x) - k) h(x - u) du$

$g(x) = \sum f(k) \rho(x,k)$ where $\rho(x,k) = \int r(m(x) - k) h(x - u) du$

Mathematical Formulation (2)

$p(x,k) = \int r(m(x) - k) h(x - t) dt$

Spatially varying resampling filter expressed in terms of output space

Mathematical Formulation (3)

• We can express $\rho(x,k)$ in terms of input space:

Let $m(u)$, we have

$\rho(x,k) = \int r(u - k) h(x - m(u)) \frac{dm}{du} du$

where $\left| \frac{dm}{du} \right|$ is the determinant of Jacobian matrix:

1D: $\frac{dm}{du} = \frac{dx}{du}$

2D: $\frac{dm}{du} = \begin{vmatrix} \frac{dx}{du} & \frac{dx}{du} \\ \frac{dy}{du} & \frac{dy}{du} \end{vmatrix}$

Wolberg: Image Processing Course Notes
Resampling Filter

\[ \rho_{\text{mag}}(x, k) = r(u) \]

Shape of \( r(u) \) remains the same, independently of \( m(u) \), (independently of scale factor)

\[ \rho_{\text{min}}(x, k) = |J|h(uJ) \]

Shape of prefilter is based on desired frequency response characteristic (performance in passband and stopband).

Unlike \( r(u) \), though, the prefilter must be scaled proportional to the minification factor (broader and shorter for more minification)

Reciprocal Relationship

\[ f(x) \leftrightarrow F(x) \]

Magnification yields less high-freq (more blurred)

Minification introduces higher freqs (more visual detail)

Intuition: \( f = 1/T \)

Consequences: narrow filters in spatial domain (desirable) yield wide frequency spectrums (undesirable). Tradeoff between accuracy and complexity.

Software (1)

```c
resample1D(IN, OUT, INlen, OUTlen, filtertype, offset)
unsigned char *IN, *OUT;
int INlen, OUTlen, filtertype, offset;
{
    int i;
    int left, right; // kernel extent in input
    int pixel; // input pixel value
    double u, x; // input (u) , output (x)
    double scale; // resampling scale factor
    double (*filter)(); // pointer to filter fct
    double fwidth; // filter width (support)
    double fscale; // filter amplitude scale
    double weight; // kernel weight
    double acc; // convolution accumulator
    scale = (double) OUTlen / INlen;
}
```

Software (2)

```c
switch(filtertype) {
    case 0: filter = boxFilter; // box filter (nearest nbr)
        fwidth = .5;
        break;
    case 1: filter = triFilter; // triangle filter (lin intrp)
        fwidth = 1;
        break;
    case 2: filter = cubicConv; // cubic convolution filter
        fwidth = 2;
        break;
    case 3: filter = lanczos3; // Lanczos3 windowed sinc fct
        fwidth = 3;
        break;
    case 4: filter = hann4; // Hann windowed sinc function
        fwidth = 4;
        break;
}
```

Software (3)

```c
if(scale < 1.0) { // minification: h(x) -> h(x*scale)*scale
    fwidth = fwidth / scale; // broaden filter
    fscale = scale; // lower amplitude
    /* roundoff fwidth to int to avoid intensity modulation */
    if(filtertype == 0) {
        fwidth = CEILING(fwidth);
    }
    else fscale = 1.0;
}
```

Fourier Transform Pairs

- The shape of \( \rho_{\text{mag}} \) is a direct consequence of the reciprocal relation between the spatial and frequency domains.

\[ H(u) = \int h(x)e^{-2\pi iux}dx \]

\[ H\left(\frac{m(u)}{s}\right) = \frac{1}{s} \int h(x)e^{-2\pi iux}dx \]

Where \( m^{-1}(x) = \frac{x}{s} \)

\[ h(au) \leftrightarrow \frac{1}{a} F\left(\frac{u}{a}\right) \]

\( a \) denotes Fourier Transform pair

Software (3)

```c
if(scale < 1.0) {
    fwidth = fwidth / scale;
    fscale = scale;
    /* roundoff fwidth to int to avoid intensity modulation */
    if(filtertype == 0) {
        fwidth = CEILING(fwidth);
    }
    else fscale = 1.0;
}
```
Software (4)

```c
/* reset acc for collecting convolution products */
acc = 0;

/* weigh input pixels around u with kernel */
for(isleft; i <= right; i++) {
    pixel = IN[(i - left) * offset];
    weight = (*filter)((u - i) * fscale);
    acc += (pixel * weight);
}

/* assign weighted accumulator to OUT */
OUT[x*offset] = acc * fscale;
```

Software (5)

```c
double boxFilter(double t)
{
    if((t > -.5) && (t <= .5)) return(1.0);
    return(0.0);
}

double triFilter(double t)
{
    if(t < 0) t = -t;
    if(t < 1.0) return(1.0 - t);
    return(0.0);
}
```

Software (6)

```c
double cubicConv(double t)
{
    double A, t2, t3;
    if(t < 0) t = -t;
    t2 = t  * t;
    t3 = t2 * t;
    A = -1.0; // user-specified free parameter
    if(t < 1.0) return((A+2)*t3 - (A+3)*t2 + 1);
    if(t < 2.0) return(A*(t3 - 5*t2 + 8*t - 4));
    return(0.0);
}
```

Software (7)

```c
double sinc(double t)
{
    t *= PI;
    if(t != 0) return(sin(t) / t);
    return(1.0);
}

double lanczos3(double t)
{
    if(t < 0) t = -t;
    if(t < 3.0) return(sinc(t) * sinc(t/3.0));
    return(0.0);
}
```

Software (8)

```c
double hann4(double t)
{
    int N = 4; // fixed filter width
    if(t < 0) t = -t;
    if(t < N)
        return(sin(t) * (.5 + .5*cos(PI*t / N)));
    return(0.0);
}
```

Image Reconstruction

Prof. George Wolberg
Dept. of Computer Science
City College or New York
Objectives

- In this lecture we describe image reconstruction:
  - Interpolation as convolution
  - Interpolation kernels for:
    - Nearest neighbor
    - Triangle filter
    - Cubic convolution
    - B-Spline interpolation
    - Windowed sinc functions

Introduction

- Reconstruction is synonymous with interpolation.
- Determine value at position lying between samples.
- Strategy: fit a continuous function through the discrete input samples and evaluate at any desired set of points.
- Sampling generates infinite bandwidth signal.
- Interpolation reconstructs signal by smoothing samples with an interpolation function (kernel).

Interpolation

For equi-spaced data, interpolation can be expressed as a convolution:

\[
f(x) = \sum_{k=0}^{K-1} c_k h(x - x_k)
\]

where \( K \) is the number of neighborhood pixels and \( c_k \) are coefficients for kernel \( h \)

Interpolation Kernel

- Set of weights applied to neighborhood pixels
- Often defined analytically
- Usually symmetric: \( h(x) = h(-x) \)
- Commonly used kernels:
  - Nearest neighbor (pixel replication)
  - Triangle filter (linear interpolation)
  - Cubic convolution (smooth; used in digital cameras)
  - Windowed sinc functions (highest quality, more costly)

Nearest Neighbor

Interpolating Polynomial: \( f(x) = f(x_k) \)

Interpolating Kernel: \( h(x) = \begin{cases} \frac{1}{2} & 0 < x < 1 \\ 0 & \text{otherwise} \end{cases} \)

Other names: box filter, sample-and-hold function, and Fourier window. Poor stopband. NN achieves magnification by pixel replication. Very blocky. Shift errors of up to 1/2 pixel are possible. Common in hardware zooms.

Triangle Filter

Interpolating Polynomial: \( f(x) = a \cdot b + c \cdot d \)

Interpolating Kernel: \( h(x) = \begin{cases} 1 & 0 < x < 1 \\ 0 & \text{otherwise} \end{cases} \)

Other names for \( h \): triangle filter, tent filter, roof function, chateau function, and Bartlett window.
Cubic Convolution (1)

Third degree approximation to sinc. Its kernel is derived from constraints imposed on the general cubic spline interpolation formula.

\[ h(x) = \begin{cases} a_0 + a_1 x + a_2 x^2 + a_3 x^3 & \text{for } |x| < 1 \\ 0 & \text{otherwise} \end{cases} \]

Determine coefficients by applying following constraints:
1. \( h(0) = 1 \) and \( h(x) = 0 \) for \( |x| = 1, 2 \)
2. \( h \) must be continuous at \( |x| = 0, 1, 2 \)
3. \( h \) must have a continuous first derivative at \( |x| = 0, 1, 2 \)

Constraint (1) states that when \( h \) is centered on an input sample, the interpolation function is independent of neighboring samples.

First 2 constraints give 4 equations:
1. \( h(0) = a_0 = 1 \)
2. \( h(1) = -4a_0 > 0 \) Concave upward at \( x = 1 \)
3. \( h(2) = 8a_0 - 4a_3 = 0 \) \( h(0) = h(2) = 0 \)

Total 7 equations, 8 unknowns = free variable \( a = a_3 \)

\[ h(x) = \begin{cases} 1 + 2(x-1)^3 & \text{for } 0 < x < 1 \\ 2 - 2(x-1)^3 & \text{for } 1 < x < 2 \\ 0 & \text{otherwise} \end{cases} \]

Cubic Convolution (2)

Cubic Convolution (3)

How to pick \( a \)? Add some heuristics (make it resemble Sinc function):
1. \( h(0) = 1 \) and \( h(x) = 0 \) for \( |x| = 1, 2 \)
2. \( h \) must be continuous at \( |x| = 0, 1, 2 \)
3. \( h \) must have a continuous first derivative at \( |x| = 0, 1, 2 \)

Common choices:
- \( a = -1 \) matches the slope of sinc at \( x = 1 \) (sharpens image)
- \( a = -0.5 \) makes the Taylor series approximately agree in as many terms as possible with the original signal
- \( a = -0.75 \) sets the second derivative of the 2 cubic polynomials in \( h \) to 1 (continuous 2nd derivative at \( x = 1 \))

Cubic Splines (1)

6 polynomial segments, each of 3rd degree. \( f_k \) are joined at \( x_k \) for \( k = 1, \ldots, n-2 \) such that \( f_k \), \( f'_k \), and \( f''_k \) are continuous.

- \( a_0 = y_0 \)
- \( a_1 = y'_0 \) where \( \Delta y_k = y_{k+1} - y_k \)
- \( a_2 = 3 \Delta y_k - 2y_{k+1} - y_k \)
- \( a_3 = -2\Delta y_k + y_{k+1} + y_k \)

(proof in App. 2)

Cubic Splines (2)

- The derivatives may be determined by solving the following system of linear equations:

\[
\begin{bmatrix}
2 & 4 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 4 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 4 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 4 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 4 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 4 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{bmatrix}
\begin{bmatrix}
\gamma_1 \\
\gamma_2 \\
\gamma_3 \\
\gamma_4 \\
\gamma_5 \\
\gamma_6 \\
\end{bmatrix}
= \begin{bmatrix}
-5y_2 + 6\gamma_2 + \gamma_3 \\
3(\gamma_2 - y_3) \\
3(\gamma_2 - y_3) \\
3(\gamma_2 - y_3) \\
-5y_2 + 6\gamma_2 + \gamma_3 \\
\end{bmatrix}
\]

- Introduce by the constraints for continuity in the first and second derivatives at the knots.

B-Splines

To analyze cubic splines, introduce cubic B-Spline interpolation kernel:

\[
k(x) = \frac{1}{2} \begin{bmatrix}
1 & 4 & 6 & 4 & 1 \\
0 & 1 & 3 & 3 & 1 \\
-2 & -4 & 3 & 3 & 1 \\
3 & 0 & -3 & -3 & 1 \\
-6 & 0 & 6 & 6 & 1 \\
0 & -6 & 15 & 15 & 1 \\
-15 & 0 & -15 & -15 & 1 \\
1 & 0 & 1 & 1 & 1 \\
\end{bmatrix} \\
\text{with } 0 \leq x < 1
\]

Parzen Window: Not interpolatory because it does not satisfy \( h(0) = 1 \) and \( h(1) = h(2) = 0 \). Indeed, it approximates the data.

Wolberg: Image Processing Course Notes
Interpolatory B-Splines

\[ f(x_k) = \sum_{i=-\infty}^{\infty} c_i w_k(x_k - x_i) \]

Since \( A(x) = \frac{1}{2}(x-1) = \frac{1}{2}(x+1) \), we have

\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 \\
4 & 4 & 4 & 4 & 4 \\
1 & 1 & 1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
c_0 \\
c_1 \\
c_2 \\
c_3 \\
c_4
\end{bmatrix}
= \begin{bmatrix}
f_0 \\
f_1 \\
f_2 \\
f_3 \\
f_4
\end{bmatrix}
\]

\[ F = K \cdot C \]

\[ C = K^{-1} \cdot F \]

\( K^{-1} \) is the inverse of the diagonal matrix; Computation is \( O(n) \).

All previous methods used data values for \( c_k \) from \( C = K^{-1} \cdot F \).

Truncated Sinc Function

- Alternative to previous kernels: use windowed sinc function.

Truncated Sinc

\[ \text{Ringing can be mitigated by using a smoothly tapering windowing function.} \]


Hann/Hamming Window

\[ \text{Hann/Hamming} = \frac{\alpha - |\alpha - \alpha| \cos \left( \frac{2\pi x}{N-1} \right)}{\alpha} \]

\( |H(f)| \) is sinc+2 shifted sincs. These cancel the right and left side lobes of \( \text{Rect}(x) \).

Blackman Window

\[ \text{Blackman} = \frac{42 + 5.4 \cos \left( \frac{2\pi x}{N-1} \right) - 2 \cos \left( \frac{6\pi x}{N-1} \right)}{42} \]

Lanczos Window (1)

\[ \text{Lanczos}(x) = \begin{cases} \sin(\pi x) & 0 < x < 2 \\ \frac{\sin(\pi x/2)}{\pi x/2} & |x| > 2 \end{cases} \]

Lanczos Window (2)

- Generalization to \( N \) lobes:

\[ \text{Lanczos}_N(x) = \begin{cases} \sin(\pi x) & 0 < x < N \\ \frac{\sin(\pi x/2)}{\pi x/2} & |x| > N \end{cases} \]

Let \( N = 3 \), this lets 3 lobes pass under the Lanczos window.

- Better passband and stopband response
Comparison of Interpolation Methods

NN, linear, cubic convolution, windowed sinc, sinc
poor ...................................................> ideal
(blocky, blurred, ringing, no artifacts)

Convolution Implementation

1. Position (center) kernel in input.
2. Evaluate kernel values at positions coinciding with neighbors.
3. Compute products of kernel values and neighbors.
4. Add products; init output pixel.

Step (1) can be simplified by incremental computation for space-invariant
warping. (newpos = oldpos + inc).
Step (2) can be simplified by LUT.

Interpolation with Coefficient Bins

Implementation #1: Interp. with Coefficient Bins (for space-invariant warps)

• Strategy: accelerate resampling by precomputing the input weights and
storing them in LUT for fast access during convolution.

\[ u \rightarrow \text{bin (quantize u)} \]

\[ h_1, h_2, h_3, \text{Kernel values} \]

\[ \text{Let } d = 1 - i/n (u \leq d < 1) \]

\[ h_1 = h(d); \quad h_{-1} = h(1-d) \]

\[ h_2 = h(1+d); \quad h_{-2} = h(2-d) \]

\[ \text{Interpolated} \]

Uninterleaved Coefficient Bins

\[ ii = \text{bin #} \]

\[ 0 \leq ii < \text{oversample} \]


\[ \text{if}(ii == 0) \text{val += IN[-3] * kern[3 * oversample - ii];} \]

Refer to code on p. 151

Kernel Position

• Since we are assuming space invariance, the new position for the kernel =
oldpos + offset.

\[ \text{Offset must be accurate to avoid accrual of error in the incremental repositioning of the kernel.} \]

Forward vs. Inverse Mapping

\[ x = X(u, v); \quad y = Y(u, v) \quad \text{Ch. 3, Sec. 1} \]

\[ u = U(x,y); \quad v = V(x, y) \]

Coefficient Bins for kernel eval for fast convolution for image reconstruction.
Fant’s Algorithm

Implementation #2: Fant’s Resampling Algorithm (for space-var. warps)

Antialiasing

Prof. George Wolberg
Dept. of Computer Science
City College or New York

Antialiasing

Point Sampling

Objectives

• In this lecture we review antialiasing:
  - Supersampling (uniform, adaptive, irregular)
  - Direct convolution
    - Feibush-Levoy-Cook
    - Gangnet-Perry-Coveignoux
    - Greene-Heckbert (Elliptical Weighted Average)
  - Prefiltering
    - Mip-maps / pyramids
    - Summed area tables

Antialiasing

• Aliasing is due to undersampling the input signal.
• It results in false artifacts, e.g., moire effects.
• Antialiasing combats aliasing. Solutions:
  • Raise sampling rate
  • Bandlimit input
  • In practice, do both in inverse mapping formulation
• Implementations:
  • Supersampling (uniform, adaptive, irregular)
  • Direct convolution
  • Prefiltering

Point Sampling

• One input sample per output pixel
• Problem: information is lost between samples
• Solution: sample more densely and average results
Supersampling (1)

- Multiple input samples per output pixel

1D:

2D:

Supersampling (2)

- 1 sample/pixel
- 4 samples/pixel
- 16 samples/pixel
- 64 samples/pixel
- 256 samples/pixel

Adaptive Supersampling

- Collect more samples in areas of high intensity variance or contrast.

If \(|f(A,B,C,D,E) > \text{thr}\) then subdivide /* quad tree */

- To avoid artifacts due to regular sampling pattern (along rectilinear grid), use irregular sampling.
- Three common forms of stochastic sampling:
  - Poisson
  - Jittered
  - Point-diffusion sampling

Direct Convolution: Feibush-Levoy-Cook (1980)

1. All input pixels contained within the bounding rect. of this quadrilateral are mapped onto output.
2. The extra selected points are eliminated by clipping them against the bounding rect. of kernel.
3. Int output pixel with weighted average of the remaining samples. Note that filter weights are stored in a LUT (e.g., a Gaussian filter.)

Advantages: 1) Easy to collect input pixels in rec. input region
2) Easy to clip output pixels in rec. output region.
3) Arbitrary filter weights can be stored in LUT.

Direct Convolution: Gangnet-Perry-Coveignoux (1982)

Improve results by supersampling.

Major axis determines supersampling rate.

They used bilinear interpolation for image reconstruction and a truncated sinc (2 pixels on each side) as kernel.

Elliptical weighted average (EWA): Greene-Heckbert (1986)

EWA distorts the circular kernel into an ellipse in the input where the weighting can be computed directly.

No mapping back to output space.

\[
Q(u,v) = Au^2 + Buv + Cv^2 \quad u = v = 0 \text{ is the center of the ellipse}
\]

\[
A = V_x^2 + V_y^2
\]

\[
B = 2(U_x V_x + U_y V_y)
\]

\[
C = U_x^2 + U_y^2
\]

where

\[
(U_x, U_y) \quad \frac{\partial}{\partial u} \quad V_x \quad V_y
\]

Point - inclusion test: \(Q(u,v) < F \) for \(F = (U_x V_x - U_y V_y)^2\)
EWA Advantages

• Only 1 function evaluated; not 4 needed for quadrilateral.
• If Q<F, then sample is weighted with appropriate LUT entry.
• In output space, LUT is indexed by r
• Instead of determining which concentric circle the point lies in the output, we determine which concentric ellipse the point lies in the input and use it to index the LUT.

Comparisons

• All direct convolution methods are O(N) where N = # input pixels accessed. In Feibush and Gangnet, these samples must be mapped into output space, EWA avoids this costly step.
• Direct convolution imposes minimal constraints on filter area (quad, ellipse) and filter kernel (precomputed LUT). Additional speedups: prefiltering with pyramids and preintegrated tables -> approx convolution integral w/ a constant # of accesses (indep. of # input pixels in preimage.)
• Drawback of pyramids and preintegrated tables: filter area must be square or rectangular; kernel must be a box filter.

Pyramids

Size of square is used to determine pyramid level to sample. Averaging or Blurring tradeoff:

d' = max [ (x-1)/x, (y-1)/y ]

d is proportional to size of the pyramid area.

Summed-Area Table

Table stores running sum

\[ \text{sum}_2 = T[x_1, y_1] - T[x_1, y_2] - T[x_2, y_1] + T[x_2, y_2] \]

Restricted to rectangular regions and box filtering

To compute current entry T[x1, y1] let sum R be current pixel value v:

\[ \begin{align*}
T[x_1, y_1] &= v + T[x_1, y_2] + T[x_2, y_1] - T[x_2, y_2] \\
\end{align*} \]

Example

<table>
<thead>
<tr>
<th>Input</th>
<th>Summed-area table</th>
</tr>
</thead>
<tbody>
<tr>
<td>90 10 20 30</td>
<td>265 460 820 1190</td>
</tr>
<tr>
<td>50 60 70 80</td>
<td>175 360 700 1040</td>
</tr>
<tr>
<td>25 75 200 180</td>
<td>125 250 520 780</td>
</tr>
<tr>
<td>100 50 70 80</td>
<td>100 150 220 300</td>
</tr>
</tbody>
</table>

Spatial Transformations

Prof. George Wolberg
Dept. of Computer Science
City College of New York
Objectives

• In this lecture we review spatial transformations:
  - Forward and inverse mappings
  - Transformations
    - Linear
    - Affine
    - Perspective
    - Bilinear
  - Inferring affine and perspective transformations

---

Forward and Inverse Mappings

• A spatial transformation defines a geometric relationship between each point in the input and output images.
• Forward mapping: \([x, y] = [X(u,v), Y(u,v)]\)
• Inverse mapping: \([u, v] = [U(x,y), V(x,y)]\)

---

Linear Transformations

\[
\begin{bmatrix}
  x \\
  y
\end{bmatrix} = \begin{bmatrix}
  a_{11} & a_{12} & 0 \\
  a_{21} & a_{22} & 0 \\
  a_{31} & a_{32} & 1
\end{bmatrix}
\]

• Above equations are linear because they satisfy the following two conditions necessary for any linear function \(L(x)\):
  1) \(L(x+y) = L(x) + L(y)\)
  2) \(L(cx) = cL(x)\) for any scalar \(c\) and position vectors \(x, y\).
• Note that linear transformation are a sum of scaled input coordinate: they do not account for simple translation.

---

Affine Transformations (1)

\[
\begin{bmatrix}
  x \\
  y
\end{bmatrix} = \begin{bmatrix}
  a_{11} & a_{12} & 0 \\
  a_{21} & a_{22} & 0 \\
  a_{31} & a_{32} & 1
\end{bmatrix}
\]

Translation:
\[
\begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix}
\]

Shear:
\[
\begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix}
\]

Scale:
\[
\begin{bmatrix}
  s_x & 0 & 0 \\
  0 & s_y & 0 \\
  0 & 0 & 1
\end{bmatrix}
\]

Rotation (\(\theta\)):
\[
\begin{bmatrix}
  \cos\theta & -\sin\theta & 0 \\
  \sin\theta & \cos\theta & 0 \\
  0 & 0 & 1
\end{bmatrix}
\]

Affine Transformations (2)

• Affine transformation have 6 degrees of freedom: \(a_{11}, a_{21}, a_{31}, a_{12}, a_{22}, a_{32}\).
• They can be inferred by giving the correspondence of three 2-D points between the input and output images. That is, 
  \[
  (u_1, v_1) \rightarrow (u_2, v_2), (u_3, v_3) \rightarrow (u_4, v_4)
  \]
  6 constraints: (3 for \(u\rightarrow x\), 3 for \(v\rightarrow y\))

• All points lie on the same plane.
• Affine transformations map triangles onto triangles.
Affine Transformations (3)

- Skew (shear)
- Rotation/Scale
- Rotation

Perspective Transformations (1)

\[
\begin{bmatrix}
    x' \\
    y' \\
    w'
\end{bmatrix} = \begin{bmatrix}
    a_{11} & a_{12} & a_{13} \\
    a_{21} & a_{22} & a_{23} \\
    a_{31} & a_{32} & a_{33}
\end{bmatrix}
\begin{bmatrix}
    u \\
    v \\
    w
\end{bmatrix} + \begin{bmatrix}
    a_{14} \\
    a_{24} \\
    a_{34}
\end{bmatrix}
\]

\[
\begin{bmatrix}
    x' \\
    y' \\
    w'
\end{bmatrix} = \begin{bmatrix}
    a_{11} & a_{12} & a_{13} & a_{14} \\
    a_{21} & a_{22} & a_{23} & a_{24} \\
    a_{31} & a_{32} & a_{33} & a_{34}
\end{bmatrix}
\begin{bmatrix}
    u \\
    v \\
    w
\end{bmatrix}
\]

- Without loss of generality, we set \(a_{33}=1\).
- This yields 8 degrees of freedom and allows us to map planar quadrilaterals to planar quadrilaterals (correspondence among 4 sets of 2-D points yields 8 coordinates).
- Perspective transformations introduces foreshortening effects.
- Straight lines are preserved.

Perspective Transformations (2)

Bilinear Transforms (1)

\[
\begin{bmatrix}
    x \\
    y
\end{bmatrix} = \begin{bmatrix}
    a_1 & b_1 \\
    a_2 & b_2
\end{bmatrix}
\begin{bmatrix}
    u \\
    v
\end{bmatrix}
\]

- 4-corner mapping among nonplanar quadrilaterals (2nd degree due to \(uv\) factors).
- Conveniently computed using separability (see p. 57-60).
- Preserves spacing along edges.
- Straight lines in the interior no longer remain straight.

Bilinear Transforms (2)

Examples

- Similarity transformation (RST)
- Affine transformation
- Perspective transformation
- Polynomial transformation
Scanline Algorithms

Prof. George Wolberg
Dept. of Computer Science
City College or New York

Objectives

• In this lecture we review scanline algorithms:
  - Incremental texture mapping
  - 2-pass Catmull-Smith algorithm
    • Rotation
    • Perspective
  - 3-pass shear transformation for rotation
  - Morphing

Catmull-Smith Algorithm

• Two-pass transform
• First pass resamples all rows: \([u, v] \rightarrow [x, v]\)
  \([x, v] = \{ F_v(u), v \} \) where \( F_v(u) = X(u, v) \) is the forward mapping fct
• Second pass resamples all columns: \([x, v] \rightarrow [x, y]\)
  \([x, y] = \{ x, G_x(v) \} \) where \( G_x(v) = Y(H_x(v), v) \)
• \( H_x(v) \) is the inverse projection of \( x' \), the column we wish to resample.
• It brings us back from \([x, v]\) to \([u, v]\) so that we can directly index into \( Y \) to get the destination \( y \) coordinates.

Example: Rotation (1)

\([x, y] = [u, v]\)

\[\begin{bmatrix}
\cos \theta & \sin \theta \\
-\sin \theta & \cos \theta
\end{bmatrix}\]

Pass 1: \([x, v] = [u \cos \theta - v \sin \theta, v]\)

Pass 2: a) Compute \( H_x(v) \). Recall that \( x = u \cos \theta - v \sin \theta \)

\[u = \frac{x + v \sin \theta}{\cos \theta}\]

b) Compute \( G_x(v) \). Substitute \( H_x(v) \) into \( y = x \sin \theta + v \cos \theta \)

\[y = \frac{x \sin \theta + v}{\cos \theta}\]

3-Pass Rotation Algorithm

• Rotation can be decomposed into two scale/shear matrices.

\[R = \begin{bmatrix}
\cos \theta & \sin \theta \\
-\sin \theta & \cos \theta
\end{bmatrix} \begin{bmatrix}
1 & \frac{\tan \theta}{\cos \theta} \\
0 & 1
\end{bmatrix}\]

• Three pass transform uses on shear matrices.

\[R = \begin{bmatrix}
\cos \theta & \sin \theta \\
-\sin \theta & \cos \theta
\end{bmatrix} \begin{bmatrix}
1 & \frac{1}{2} \\
0 & 1
\end{bmatrix} \begin{bmatrix}
\cos \theta & \sin \theta \\
-\sin \theta & \cos \theta
\end{bmatrix} \begin{bmatrix}
1 & 0 \\
\frac{1}{2} & 1
\end{bmatrix}\]

• Advantage of 3-pass: no scaling necessary in any pass.
3-Pass Rotation

Software Implementation

• The following slides contain code for initMatrix.c to produce a 3x3 perspective transformation matrix from a list of four corresponding points (e.g. image corners).
• That matrix is then used in perspective.c to resample the image.
• The code in resample.c performs the actual scanline resample.

initMatrix.c (1)

```c
/* initMatrix:
 *
 * Given Icorr, a list of the 4 correspondence points for the corners
 * of image I, compute the 3x3 perspective matrix in Imatrix.
 *
 * void initMatrix(imageP I, imageP Icorr, imageP Imatrix)
 *
 * init pointers */
 * a = (float *) Imatrix->buf[0];
 * p = (float *) Icorr->buf[0];
 */

int w, h;
float *p, *a, a13, a23;
float y0, y1, y2, y3;
float dx1, dx2, dx3, dy1, dy2, dy3;

/* init pointers */
*a = (float *) Imatrix->buf[0];
*p = (float *) Icorr->buf[0];
/* init u,v,x,y vars and print them */
x0 = *p++;
y0 = *p++;
x1 = *p++;
y1 = *p++;
x2 = *p++;
y2 = *p++;
x3 = *p++;
y3 = *p++;

/* compute auxiliary vars */
dx1 = x1 - x2;
dx2 = x3 - x2;
dx3 = x0 - x1 + x2 - x3;
dy1 = y1 - y2;
dy2 = y3 - y2;
dy3 = y0 - y1 + y2 - y3;
```

initMatrix.c (2)

```c
w = I->width;
h = I->height;

UI_printf("Correspondence points:
%4d %4d %6.1f %6.1f\n", 0, 0, x0, y0);
UI_printf("%4d %4d %6.1f %6.1f\n", w, 0, x1, y1);
UI_printf("%4d %4d %6.1f %6.1f\n", w, h, x2, y2);
UI_printf("%4d %4d %6.1f %6.1f\n", 0, h, x3, y3);

/* compute 3x3 transformation matrix:
 * a0 a1 a2
 * a3 a4 a5
 * a6 a7 a8
 */
a13  = (dx3*dy2 - dx2*dy3) / (dx1*dy2 - dx2*dy1);
a23  = (dx1*dy3 - dx3*dy1) / (dx1*dy2 - dx2*dy1);
a[0] = (x1-x0+a13*x1) / w;
a[1] = (y1-y0+a13*y1) / w;
a[2] = a13 / w;
a[3] = (x3-x0+a23*x3) / h;
a[4] = (y3-y0+a23*y3) / h;
a[5] = a23 / h;
a[6] = x0;
a[7] = y0;
a[8] = 1;
```

initMatrix.c (3)

```c
/* compute 3x3 transformation matrix:
 * a0 a1 a2
 * a3 a4 a5
 * a6 a7 a8
 */
a13 = (dx3*dy2 - dx2*dy3) / (dx1*dy2 - dx2*dy1);
a23 = (dx1*dy3 - dx3*dy1) / (dx1*dy2 - dx2*dy1);
a[0] = (x1-x0+a13*x1) / w;
a[1] = (y1-y0+a13*y1) / w;
a[2] = a13 / w;
a[3] = (x3-x0+a23*x3) / h;
a[4] = (y3-y0+a23*y3) / h;
a[5] = a23 / h;
a[6] = x0;
a[7] = y0;
a[8] = 1;
```

perspective.c (1)

```c

/* perspective:
 *
 * Apply a perspective image transformation on input I1.
 * The 3x3 perspective matrix is given in Imatrix.
 * The output is stored in I2.
 *
 * void perspective(imageP I1, imageP Imatrix, imageP I2)
 *
 * init 1, w, h, wv, hh;
 * u = (float) *I1->buf[0];
 * v = (float) *I1->buf[1];
 */

int i, w, h, wv, hh;
uchar *p1, *p2;
float u, v, x, y, xmin, xmax, ymin, ymax, *a, *p;
imageP II;
w = I1->width;
h = I1->height;
a = (float *) Imatrix->buf[0];
```
perspective.c (2)

```c
xmin = xmax = X(a, 0, 0);
x = X(a, w, 0); xmin = MIN(xmin, x); xmax = MAX(xmax, x);
x = X(a, 0, h); xmin = MIN(xmin, x); xmax = MAX(xmax, x);

ymin = ymax = Y(a, 0, 0);
y = Y(a, w, 0); ymin = MIN(ymin, y); ymax = MAX(ymax, y);
y = Y(a, w, h); ymin = MIN(ymin, y); ymax = MAX(ymax, y);

ww = CEILING(xmax) - FLOOR(xmin);
hh = CEILING(ymax) - FLOOR(ymin);

/* allocate mapping fct buffer */
x = MAX(MAX(w, h), MAX(ww, hh));
F = (float *) malloc(x * sizeof(float));
if(F == NULL) IP_bailout("perspective: No memory");
/* allocate intermediate image */
II = IP_allocImage(ww, h, BW_TYPE);
IP_clearImage(II);
p1 = (uchar *) I1->buf[0];
p2 = (uchar *) II->buf[0];
```

perspective.c (3)

```c
/* first pass: resample rows */
for(v=0; v<h; v++) {
    /* init forward mapping function F; map xmin to 0 */
    for(u=0; u<w; u++) F[(int) u] = X(a, u, v) - xmin;
    resample(p1, w, 1, F, p2);
    p1 += w;
p2 += ww;
}
/* display intermediate image */
IP_copyImage(II, NextImageP);
IP_displayImage();
/* init final image */
IP_copyImageHeader(I1, I2);
I2->width  = ww;
I2->height = hh;
IP_initChannels(I2, BW_TYPE);
IP_clearImage(I2);
```

resample.c (1)

```c
/* Resample the len elements of src (with stride offst) into dst according *
to the monotonic spatial mapping given in F (len entries). *
The streamline is assumed to have been cleared earlier. */
void resample(char *src, int len, int offst, float *F, char *dst) {
    int u, uu, x, xx, ix0, ix1, I0, I1, pos;
    double x0, x1, dI;
    if(F[0] < F[len-1])
        pos = 1; /* positive output stride */
    else pos = 0; /* negative output stride */
    for(u=0; u<len-1; u++) {
        /* index into src */
        uu = u * offst;
        /* output interval (real and int) for input pixel u */
        if(pos) {/* positive stride */
            ix0 = x0 = F[u];
            ix1 = x1 = F[u+1];
            I0  = src[uu];
            I1  = src[uu+offst];
        } else { /* flip interval to enforce positive stride */
            ix0 = x0 = F[u+1];
            ix1 = x1 = F[u];
            I0  = src[uu+offst];
            I1  = src[uu];
        }
        /* index into dst */
        xx = ix0 * offst;
        /* check if interval is embedded in one output pixel */
        if(ix0 == ix1) {
            dst[xx] += I0 * (x1-x0);
            continue;
        }
        /* central interval */
        xx += offst;
        dI = (I1-I0) / (x1-x0);
        for(x=ix0+1; x<ix1; x++,xx+=offst)
            dst[xx] = I0 + dI*(x-x0);
        /* right straddle */
        if(x1 != ix1)
            dst[xx] += (I0 + dI*(ix1-x0)) * (x1-ix1);
    }
```

resample.c (2)

```c
else { /* flip interval to enforce positive stride */
    ix0 = x0 = F[u+1];
    ix1 = x1 = F[u];
    I0  = src[uu+offst];
    I1  = src[uu];
}
/* index into dst */
xx = ix0 * offst;
/* check if interval is embedded in one output pixel */
if(ix0 == ix1) {
    dst[xx] += I0 * (x1-x0);
    continue;
}
/* right straddle */
if(ix1 != ix0)
    dst[xx] += (I0 + dI*(ix1-x0)) * (x1-ix1);
```

resample.c (3)

```c
/* left straddle */
dst[xx] += I0 * (xx-ix0);
```
Objectives

In this lecture we review digital image warping:
- Geometric transformations
- Forward inverse mapping
- Sampling
- Image reconstruction
- Interpolation kernels
- Separable transforms
- Fant’s resampling algorithm

Definition

Image warping deals with the geometric transformation of digital images.

Geometric Transformations

- Affine
- Perspective
- Bilinear
- Polynomial
- Splines
- Elastic (local deformations)

Spatial Transformations

- Forward Mapping
  \([x, y] = [X(u, v), Y(u, v)]\)
- Inverse Mapping
  \([u, v] = [U(x, y), V(x, y)]\)

Forward / Inverse Mapping
Sampling

- Point Sampling

- Area Sampling

Area Sampling

- Treats pixels as finite areas
- Avoids aliasing (undersampling) artifacts
- Approximated by supersampling

Supersampling

- Average of projected subpixels

Image Reconstruction

- Pixel values are known at integer positions
- Samples can project to real-valued positions
- How do we evaluate the image values at these real-valued positions? Reconstruction

Interpolation

- Reconstruction interpolates the input
- In practice, interpolation is performed at points of interest only, not entire function
- Interpolation is achieved by convolution

Convolution

\[ g(x) = \sum_{k=0}^{N} f(x_k) h(x-x_k) \]
Interpolation Functions

Interpolation functions/kernels include:
- Box filter
- Triangle filter
- Cubic convolution
- Windowed sinc functions

Box Filter

- Nearest neighbor interpolation
- Blocky artifacts may occur

Triangle Filter

- Linear interpolation
- Popular for use with small deformations

Cubic Convolution

- Local cubic interpolation algorithm
- Advanced feature in digital cameras

Windowed Sinc Function

- Smoothly tapered ideal sinc function

Inverse Mapping

- Visit output in scanline order
- Supersampling approximates area sampling
- Popular in computer graphics
Forward Mapping

- Visit input in scanline order
- Use output accumulator array
- 2D antialiasing is difficult
- Separable transforms facilitate efficient solution

Separable Transforms

\[ [X(u, v), Y(u, v)] = F(u, v) \circ G(x, v) \]

- \( F(u, v) \) is a row-preserving transformation that maps all input points to their final column positions, i.e., \([x, v]\).
- \( G(x, v) \) is a column-preserving transformation that maps the \([x, v]\) points to their final row positions, i.e., \([x, y]\).

Catmull-Smith Algorithm

- **First pass**
  Maps image \( S(u, v) \) into intermediate image \( I(x, v) \)
  \[ I(x, v) = S(X(u, v), v) \]
- **Second pass**
  Maps \( I(x, v) \) into target image \( T(x, y) \)
  \[ T(x, y) = I(x, Y(H_x(v), v)) \]
  where \( H_x \) is the solution to \( x = X(u, v) \) for \( u \)

2-Pass Rotation

\( F \)

\( G \)

\( X, Y \)

2-Pass Perspective

\( F \)

\( G \)

\( X, Y \)

Fant’s Algorithm

- Forward mapping intensity resampling
- Scanline order in input and output
- Amenable to hardware implementation
Fant's algorithm: Example (1)

\[ \begin{array}{c|ccccc} \hline XLUT & 6 & 2.3 & 3.2 & 3.3 & 3.9 \\ \hline YLUT & 100 & 106 & 115 & 120 \\ \hline I & 100 & 106 & 92 & 90 \\ \hline YLUT_x & 100 & 101 & 105 & 113 \\ \hline I_x & 40 & 101 & 106 & 82 \\ \hline \end{array} \]

Fant's algorithm: Example (2)

\[ I_y(0) = (100)(.4) = 40 \]
\[ I_y(1) = \left[ (100) \left[ 1 - \frac{4}{17} \right] + (106) \left[ \frac{4}{17} \right] \right] (1) = 101 \]
\[ I_y(2) = \left[ (100) \left[ 1 - \frac{14}{17} \right] + (106) \left[ \frac{14}{17} \right] \right] (1.3) + (106)(.7) = 106 \]
\[ I_y(3) = \left[ (106) \left[ 1 - \frac{7}{9} \right] + (92) \left[ \frac{7}{9} \right] \right] (1.7) + (92)(1) + (90)(.6) = 82 \]

Bibliography


Image Morphing

Prof. George Wolberg
Dept. of Computer Science
City College or New York

Objectives

- In this lecture we review digital image morphing, a powerful visual effect for creating a fluid transformation from one image to another.
  - Background
  - Morphing Algorithms
    - Mesh (parametric grid)
    - Line pairs
    - Scattered points
  - Examples

Inbetween Image Generation

Cross-dissolve vs. Image Morphing
Mesh Warp (1)

Source / Target Meshes

Mesh Warp (3)
Intermediate Mesh

Mesh Warp (4)
Horizontal Resampling

Mesh Warp (5)
Vertical Resampling

Line Warp (1)

\[
\begin{align*}
\mathbf{u} &= \frac{(\mathbf{X} - \mathbf{P}) \cdot (\mathbf{Q} - \mathbf{P})}{\|\mathbf{Q} - \mathbf{P}\|^2} \\
\mathbf{y} &= \frac{(\mathbf{X} - \mathbf{P}) \cdot \text{Perp}(\mathbf{Q} - \mathbf{P})}{\|\mathbf{Q} - \mathbf{P}\|^2} \\
\mathbf{X}' &= \mathbf{P}' + \mathbf{u} \cdot (\mathbf{Q}' - \mathbf{P}') + \mathbf{y} \cdot \text{Perp}(\mathbf{Q}' - \mathbf{P}')
\end{align*}
\]
Line Warp (2)

Destination Image

Source Image

Scattered Point Constraints
Treat correspondence data as scattered data:
\([x, y] = [X(u, v), Y(u, v)]\)

Example

Uniform Transition

Nonuniform Transition

Morph Sequence
Morph Sequence

Bibliography


Objectives

- In this lecture we review image registration, a process to align multiple images together.
  - Affine registration
  - Perspective registration
  - Polar coordinates
  - Log-polar algorithm
  - Examples

Introduction

- Image registration refers to the problem of aligning a set of images.
- The images may be taken at different times, by different sensors, or from different viewpoints.

Applications

- Multisensor data fusion
  - integrating information taken from different sensors
- Image analysis / surveillance / change detection
  - for images taken at different times/conditions
- Image mosaics
  - generating large panoramas from overlapping images
- Super-resolution images / noise removal
  - integrating multiple registered images
Geometric Transformations

Geometric transformation models for registration:

- Rigid transformation
  - translation, rotation (3 parameters)
- Affine transformation
  - translation, rotation, scale, shear (6 parameters)
- Perspective transformation
  - affine, perspective foreshortening (8 parameters)
- Local transformation
  - terrain relief, nonlinear, nonrigid

Parameter Estimation (1)

Moderate perspective
\(|\beta, \gamma| < 40^\circ|

Severe perspective
\(40^\circ < |\beta, \gamma| < 90^\circ|

Parameter Estimation (2)

The objective function (similarity measure):

\[
\chi^2(a) = \sum_{i=1}^{N} (I_i(x,y) - L_2(Q_i(x',y')))^2
\]

where \(Q_i\) is the affine or perspective transform applied to image \(I_2\). We use the Levenberg-Marquardt method to minimize the above objective function for each pyramid level.

Affine Registration

Image \(I_1\)

Image \(I_2\)

\(I_2\) registered to \(I_1\)

Actual parameters: \(RST = (30^\circ, 1.5, 30, 30)\)

Estimated parameters: \(RST = (29.99^\circ, 1.51, 29.51, 30.66)\)

Local Affine Approximation

Piecewise Affine Approximation

- Consider \(I_2\) subdivided into a regular grid of tiles
- For each tile \(T_2\) search for similar tile \(T_1\) in \(I_1\) by performing affine registration using LMA
- Get collection of affine parameters and tile centers
- Plug into system of equations
- Solve for 8 persp. parameters using the pseudoinverse
Example (1)

- Perspective Registration
  - Video mosaic

Example (2)

- Glare Removal
  - Video stabilization
Discussion

- Method: Levenberg-Marquadt Alg. (nonlinear least squares)
- Benefit: estimation of real-valued parameters
- Drawback: the registered images must be fairly close in scale (<1.5), rotation (<45°), and translation
- Solution: log-polar registration

Polar Coordinates

Radial lines in \((x, y)\) Cartesian space map to horizontal lines in \((r, \theta)\) polar coordinate space.

\[
\begin{align*}
r &= \sqrt{(x - x')^2 + (y - y')^2} \\
\theta &= \tan^{-1} \frac{y - y'}{x - x'}
\end{align*}
\]

Log-Polar Coordinates

Radial lines in \((x, y)\) map to horizontal lines in \((\log r, \theta)\) polar coordinate space.

Log-Polar Registration

- Benefit: Correlation recovers rotation/scale (fast)
- Drawback: Image origins must be known
- Solution:
  1. Crop central region \(I_c\) from \(I_i\)
  2. Compute \(I_{pc}\), the log-polar transformation of \(I_c\)
  3. For all positions \((x, y)\) in \(I_i\):
     - Crop region \(I_c\)
     - Compute \(I_{pc}\)
     - Cross-correlate \(I_{pc}\) and \(I_{pc}\) to find \((dx, dy)\)
     - if maximum correlation, save \((x, y)\) and \((dx, dy)\)
  4. Scale \(\rightarrow dx\)
  5. Rotation \(\rightarrow dy\)
  6. Translation \(\rightarrow (x, y)\)
Biological Motivations (1)

Despite the difficulties of nonlinear processing, the log-polar transform has received considerable attention. [Marshal41] et. al. and [Hubel74] discovered a log-polar mapping in the primate visual system and this conformal map is an accepted model of the representation of the retina in the primary visual cortex in primates [Schwartz79] [Weinshall87].

Biological Motivations (2)

The left figure shows contours of cone cell density. In fact, the density of the ganglion cells which transmit information out of the retina is distributed logarithmically. From Osterberg, G. (1935)

**Benefits:**
1. Reduce the amount of information traversing the optical nerve while maintaining high resolution in the fovea and capturing a wide field of view in the periphery.
2. Invariant to scale and rotation.
Related Work

- The Fourier-Mellin transform uses log-polar mapping to align images related by scale, rotation, and translation [Casasent76, Decastro87, Chan96, Reddy96, Lucchese97, Chang97, Luochse97a, Luochse97b, Stone97, Stone97, Keller97].
- Detect lines in log-Hough space [Weiman79, Giesler98, Weiman90, Young00].
- Recognizing 2D objects that were arbitrarily rotated or scaled [Sandini92, Ferrari95].
- Tracking a moving object [Capurro97].
- Estimation of time-to-impact from optical flow [Sandini91].
- Finding disparity map in stereo images [Sandini01].
- Using log-polar transform to calculate time-to-crash for mobile vehicles [Pardo02].
- A foveated binocular stereo system using log polar transforms [Bernardino02].

Previous work: Fourier-Mellin

- Example: \( s = 1.5, \theta = 36^\circ \)

Previous work: Fourier-Mellin

- Benefit: We can search for the scale and rotation independent of the translations.
- Drawbacks: Large translation, scale, or mild perspective will alter the coefficients of the finite and discrete Fourier transform.

Example from INRIA
Example (1)

Example (2)

Example (3)

Example: Large Perspective (1)

Example: Large Perspective (2)

Summary

- Log-polar registration:
  - recovers large-scale rotation/scale/translation
  - applies fast correlation in log-polar space
  - exploits fast multiresolution processing
  - initial estimate for affine/perspective registration

- Perspective registration:
  - handles small distortions with subpixel accuracy
  - applies Levenberg-Marquardt algorithm
  - nonlinear least squares optimization
  - exploits fast multiresolution processing