M. D. Adams, *Lecture Slides for Multiresolution Signal and Geometry Processing (Version 2015-02-03)*, University of Victoria, Victoria, BC, Canada, Feb. 2015, xi+587 slides, ISBN 978-1-55058-535-3 (paperback), ISBN 978-1-55058-536-0 (PDF).

Please Show Your Support for These Lecture Slides

If you like these lecture slides, please show your support for them by doing one or more of the following:

1. Post a review of the lecture slides on Google Play Books and/or Google Books. **This is, by far, the most helpful thing that you can do.**
2. Rate the lecture slides on Google Play Books and/or Google Books.
3. Give a +1 to the lecture slides on Google Play Books and/or Google Books.

For your convenience, the URLs for the lecture slides on both Google Play Books and Google Books are given below. Each URL is also given in the form of a QR code.

- The lecture slides on Google Play Books:
  https://play.google.com/store/books/details?id=QpGBBgAAQBAJ
- The lecture slides on Google Books:
  http://books.google.com/books?id=QpGBBgAAQBAJ

This page is intentionally left blank.

Lecture Slides for
Multiresolution Signal and Geometry Processing

(Version: 2015-02-03)

Michael D. Adams

Department of Electrical and Computer Engineering
University of Victoria, Victoria, BC, Canada

University of Victoria

These lecture slides are for the textbook:
M. D. Adams, *Multiresolution Signal and Geometry Processing: Filter Banks, Wavelets, and Subdivision (Version: 2013-09-26)*, University of Victoria, Victoria, BC, Canada, 2013. ISBN 978-1-55058-508-7 (PDF). ISBN 978-1-55058-507-0 (paperback).

To download a ***free*** electronic copy of this textbook or for additional information and resources related to this textbook, please visit:
`http://www.ece.uvic.ca/~mdadams/waveletbook`

To join the Google Group for this textbook, please visit:
`http://groups.google.com/group/waveletbook`

The author has taken care in the preparation of this document, but makes no expressed or implied warranty of any kind and assumes no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Published by the University of Victoria, Victoria, BC, Canada

Photography by Michael Adams

This document was typeset with LaTeX.

ISBN 978-1-55058-535-3 (paperback)
ISBN 978-1-55058-536-0 (PDF)

License I

Creative Commons Legal Code

Attribution-NonCommercial-NoDerivs 3.0 Unported

> CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

a. "Adaptation" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

b. "Collection" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or

License II

broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

c. "Distribute" means to make available to the public the original and copies of the Work through sale or other transfer of ownership.

d. "Licensor" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

e. "Original Author" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

f. "Work" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

License III

g. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

h. "Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

i. "Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections; and,

b. to Distribute and Publicly Perform the Work including as incorporated in Collections.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make

License IV

Adaptations. Subject to 8(f), all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Section 4(d).

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested.

b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.

c. If You Distribute, or Publicly Perform the Work or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for

License V

attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Collection, at a minimum such credit will appear, if a credit for all contributing authors of Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

d. For the avoidance of doubt:

   i. Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

   ii. Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,

   iii. Voluntary License Schemes. The Licensor reserves the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License that is for a

License VI

```
      purpose or use which is otherwise than noncommercial as permitted
      under Section 4(b).
 e. Except as otherwise agreed in writing by the Licensor or as may be
    otherwise permitted by applicable law, if You Reproduce, Distribute or
    Publicly Perform the Work either by itself or as part of any
    Collections, You must not distort, mutilate, modify or take other
    derogatory action in relation to the Work which would be prejudicial
    to the Original Author's honor or reputation.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR
OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY
KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE,
INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY,
FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF
LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS,
WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION
OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE
LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR
ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES
ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS
BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

 a. This License and the rights granted hereunder will terminate
    automatically upon any breach by You of the terms of this License.
    Individuals or entities who have received Collections from You under
    this License, however, will not have their licenses terminated
    provided such individuals or entities remain in full compliance with
    those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any
    termination of this License.
 b. Subject to the above terms and conditions, the license granted here is
    perpetual (for the duration of the applicable copyright in the Work).
    Notwithstanding the above, Licensor reserves the right to release the
    Work under different license terms or to stop distributing the Work at
    any time; provided, however that any such election will not serve to
    withdraw this License (or any other license that has been, or is
```

License VII

required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

d. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

e. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

License VIII

Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.

Creative Commons may be contacted at http://creativecommons.org/.

Acknowledgments

- The author would like to thank the groups/organizations that provided some of the datasets used in the preparation of these slides.
- In particular, several 3-D models were provided courtesy of the Stanford Computer Graphics Laboratory (e.g., Bunny and Armadillo) and NASA (e.g., telescope dish, spacesuit, and base station).

Part 1

Overview

Course Overview

- course is about two-thirds traditional digital signal processing and one-third digital geometry processing (which can be viewed as extension of traditional digital signal processing to handle geometric objects)
- digital geometry processing has close ties to computer graphics and computational geometry
- three main topics covered by course:
  1. multirate systems (e.g., filter banks and transmultiplexers)
  2. wavelet systems
  3. subdivision surfaces and subdivision wavelets
- supplemental topics covered by course, relating to applications:
  - C++ programming language
  - computer graphics (e.g., OpenGL)
  - computational geometry (e.g., CGAL, polygon meshes)
- some additional mathematical background introduced to allow deeper understanding of course material (e.g., topics from functional analysis, Fourier analysis, geometry, topology)
- covers theory, practical issues, and applications

Why Software?

- software is pervasive
- expertise in software becoming essential for successful career in engineering
- software not just for computer science majors anymore
- strong background in software greatly improves chances of finding employment
- applies to both research and non-research jobs
- applies to both jobs in industry (e.g., software designer) and academia (e.g., professor)

Why C++?

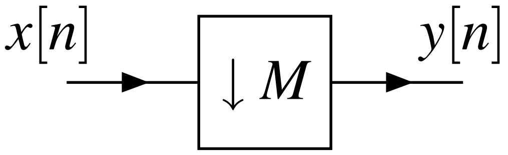
- general purpose
- international standard, vendor neutral
- efficient
- supported on many platforms
- many jobs require knowledge of C++
- superset of C (two languages for price of one)
- likely to continue to be dominant language into future (built on top of C which is still going strong after 40 years)
- many other languages inspired by C++
- easier to migrate from C++ to C, Java, and many other languages than other way around

Multirate Systems

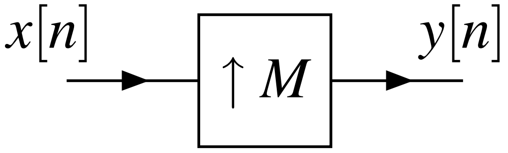
- unirate system employs single sampling rate
- multirate system uses multiple sampling rates
- sometimes multirate system can perform task more easily or efficiently than possible with unirate system
- task may fundamentally involve signals sampled at different rates, making use of multirate system unavoidable

Basic Multirate Building Blocks

- multirate systems have all of same building blocks as unirate systems plus two new ones:
  1. downsampler
  2. upsampler
- M-fold downsampler: decreases sampling rate by integer factor M

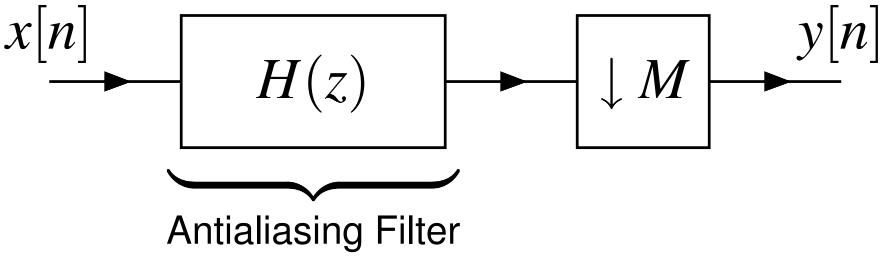


- M-fold upsampler: increases sampling rate by integer factor M

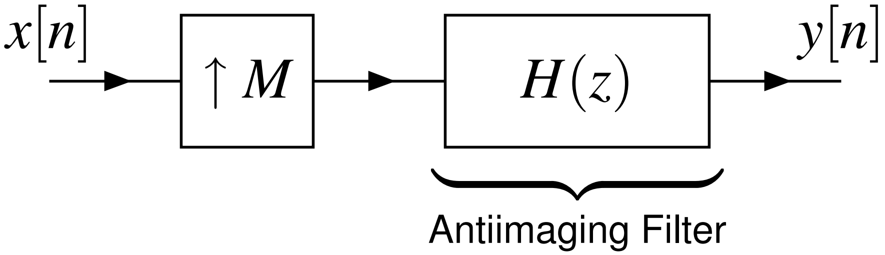


Sampling Rate Converters

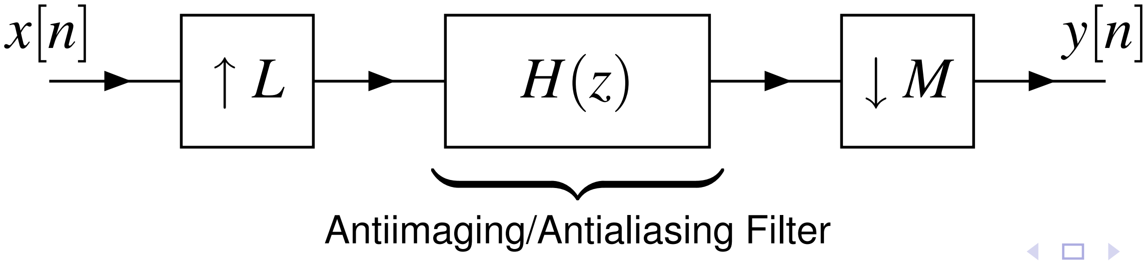
- M-fold decimator: decrease rate by integer factor M without aliasing



- M-fold interpolator: increase rate by integer factor M without imaging



- rational sampling rate converter (cascade of L-fold interpolator and M-fold decimator): change rate by factor L/M without aliasing/imaging

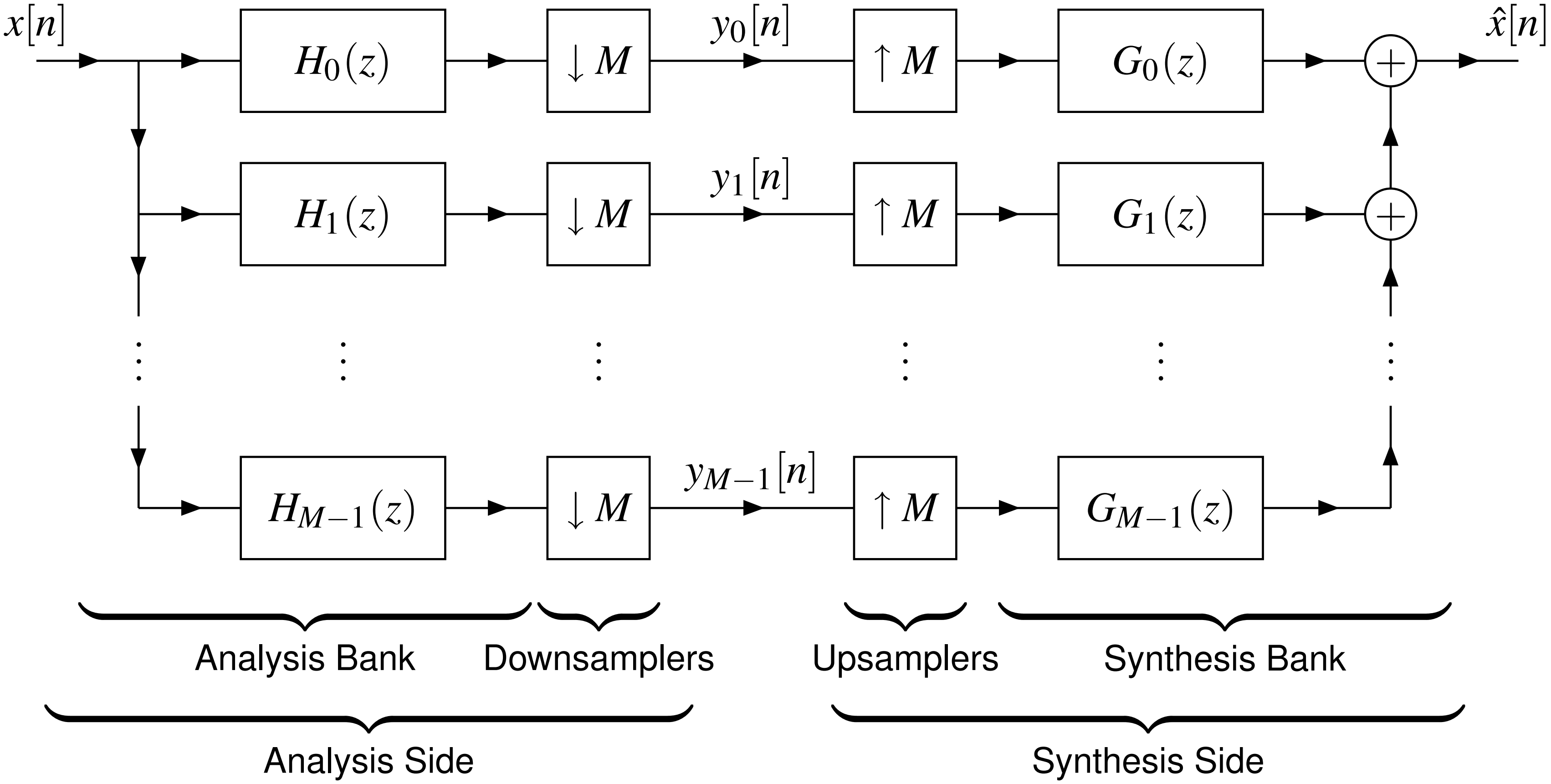


Use of Sampling Rate Converters

- sampling rate converters (i.e., decimators, interpolators) used pervasively in multirate systems
- in many applications, need to convert between different sampling rates
- streaming video/audio at different rates
- many different sampling rates commonly used for audio/music/voice data:
  - studio recording: 44.1 kHz, 48 kHz, 88.2 kHz, 96 kHz, 192 kHz
  - MPEG-1 Audio Layer 3 (MP3): 44.1 kHz (typical), 32 kHz, 48 kHz
  - Digital Audio Tape (DAT): 48 kHz (typical), 44.1 kHz, 32 kHz
  - Compact Disc (CD): 44.1 kHz
  - DVD Audio: 44.1 kHz, 192 kHz
  - GSM-FR: 8 kHz

Filter Banks

- M-channel uniformly maximally decimated (UMD) filter bank



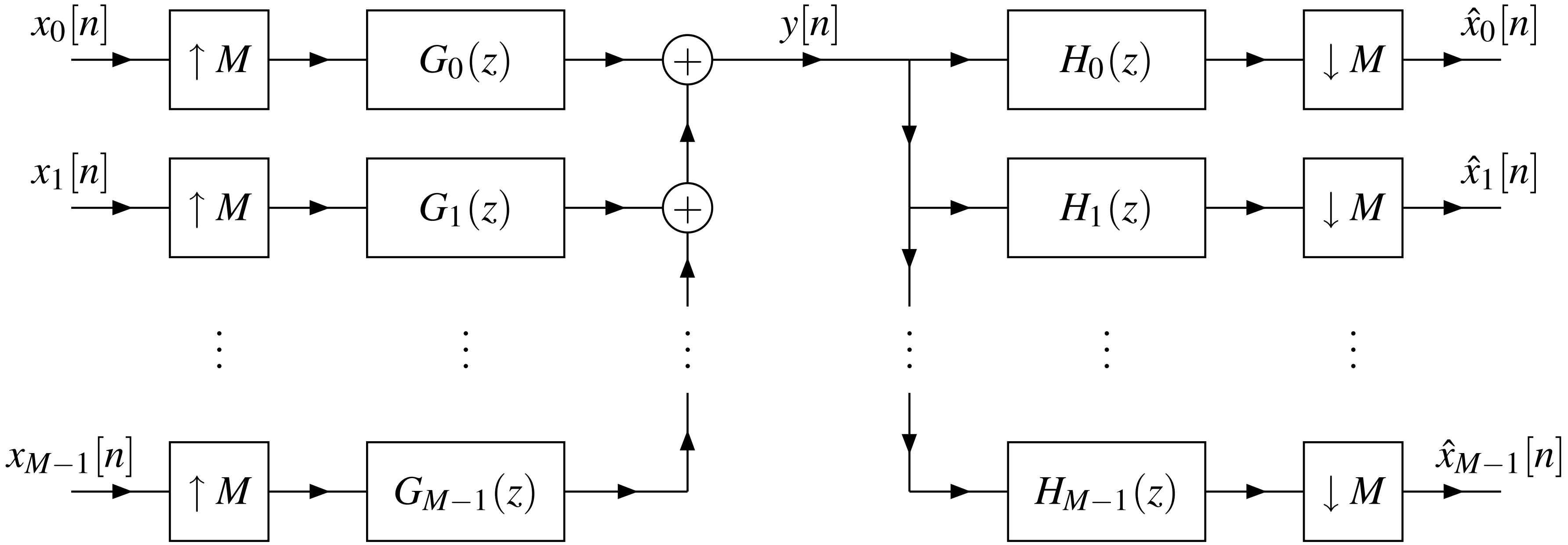
- very useful in many applications

Applications of Filter Banks

- signal denoising, restoration, and enhancement
- data compression (e.g., speech, audio, image, video, ECG, and so on)
- adaptive filtering (e.g., inverse filtering, room acoustics modelling, echo cancellation, equalization)
- data encryption
- error control coding

Transmultiplexers

- M-channel transmultiplexer



- dual structure to filter bank
- very useful in many applications

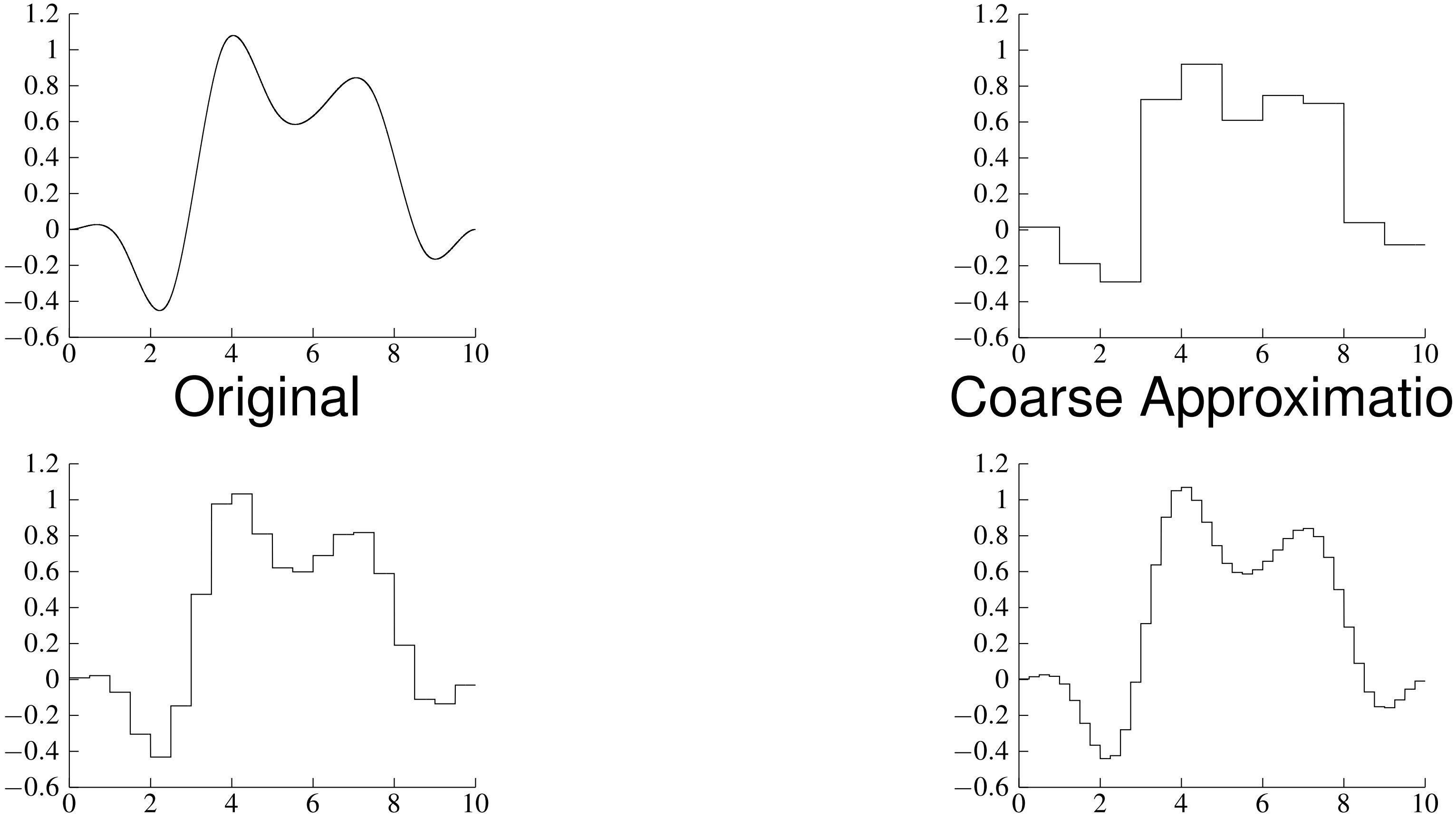
Applications of Transmultiplexers

- used pervasively in communication systems
- frequency-division multiple access (FDMA) systems, including orthogonal frequency-division multiplexing (OFDM) systems (e.g., 802.11 a/g/n)
- time-division multiple access (TDMA) systems (e.g., GSM)
- code-division multiple access (CDMA) systems (e.g., CDMA2000)
- multicarrier modulation, such as in asynchronous digital subscriber line (ADSL) systems

Wavelet Systems

- basis representation for signals/functions just like Fourier series
- function x represented in terms of basis $\{\varphi_n\}$ as $x(t) = \sum_n a_n \varphi_n(t)$ (e.g., Fourier series uses $\varphi_n(t) = e^{jn\omega_0 t}$)
- just one Fourier basis, but many wavelet bases
- basis associated with wavelet system has particular mathematical structure
- many variants of wavelet systems (e.g., univariate/multivariate, dyadic/M-adic, bounded/unbounded domains, subdivision surfaces, and so on)
- represent function in terms of information at different resolutions or scales (e.g., coarse approximation with large-scale features but little detail, or fine approximation with considerable detail)
- wavelet systems closely related to filter banks

Approximations Using Haar Wavelet System



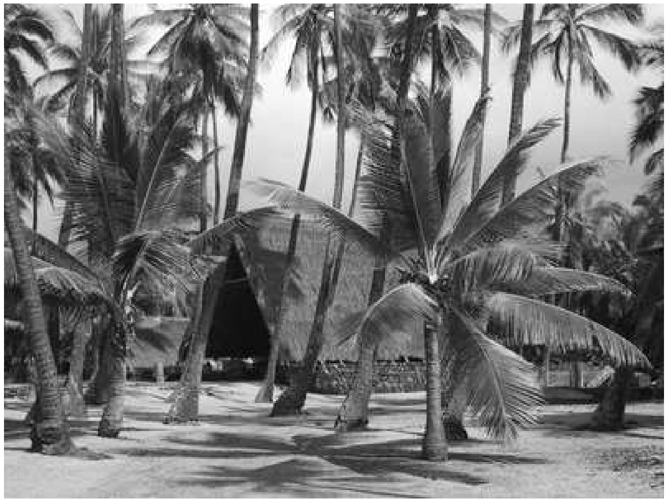
Original

Coarse Approximation

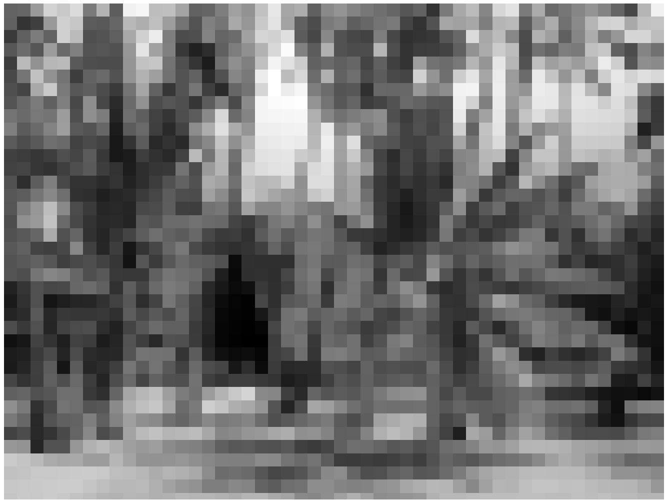
Medium Approximation

Fine Approximation

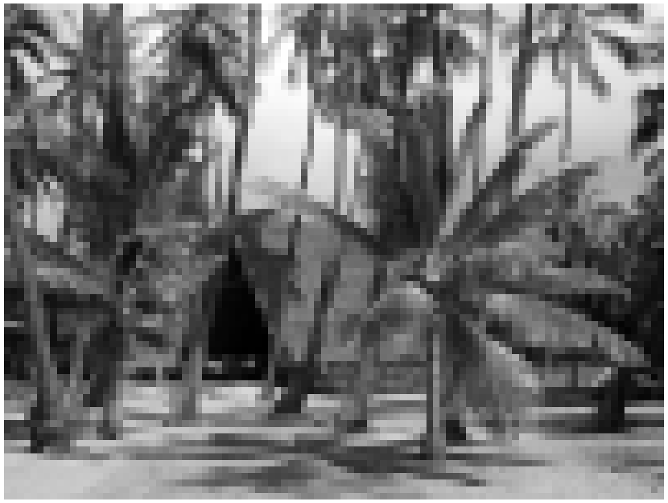
Approximations Using Haar Wavelet System: 2-D Example



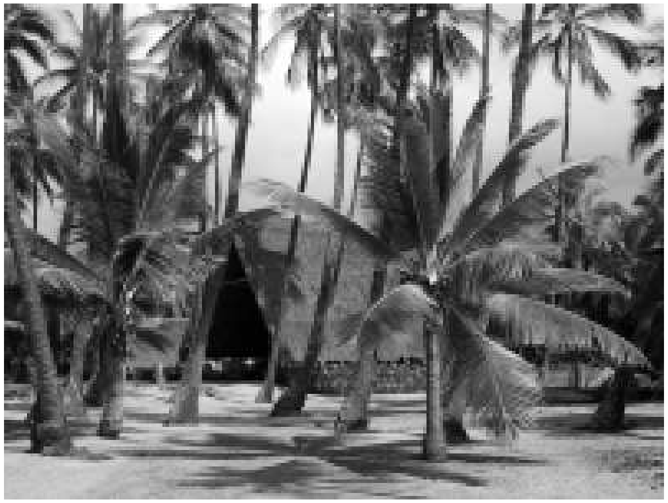
Original



Coarse Approximation



Medium Approximation



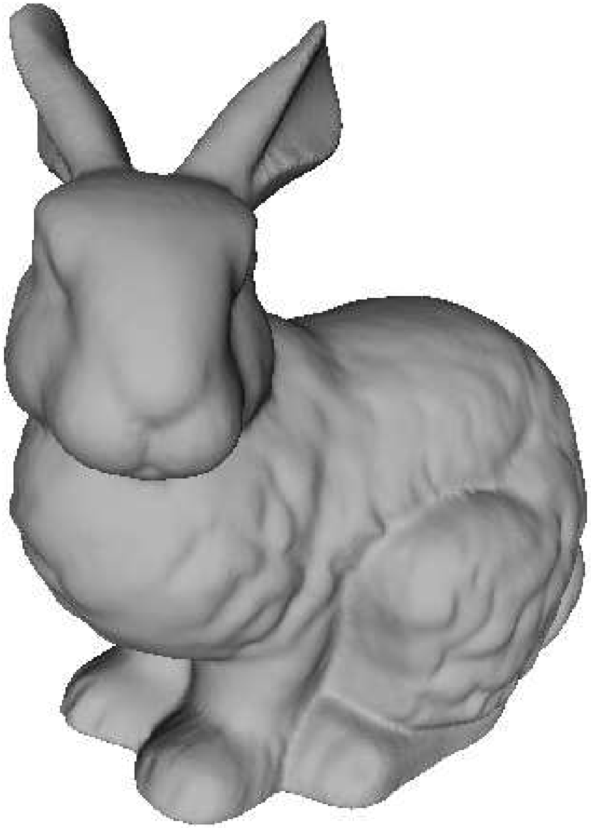
Fine Approximation

Applications of Wavelet Systems

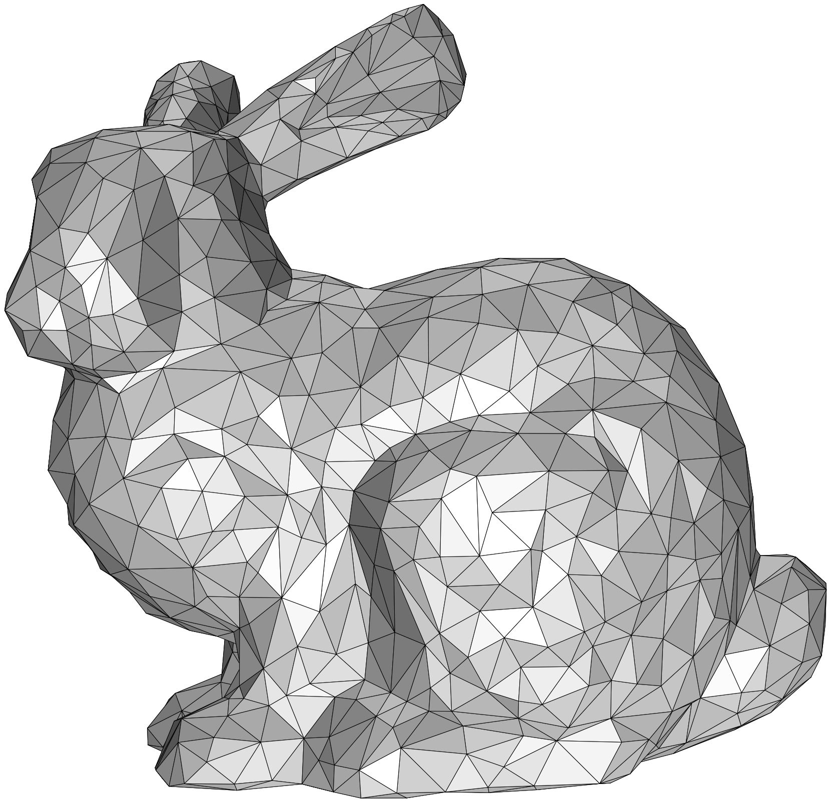
- signal compression (e.g., image, video, audio, ECG, volumetric data, light fields)
- signal denoising, restoration, and enhancement
- numerical analysis (e.g., solution of ordinary/partial differential equations)
- nonstationary signal analysis, singularity detection
- pattern recognition, feature extraction, texture classification, fault detection
- data encryption
- error control coding
- quantum mechanical modelling

Digital Geometry Processing

- digital geometry processing deals with representation and manipulation of geometric objects such as surfaces and polygon meshes

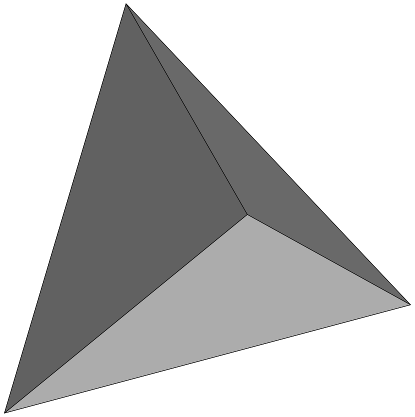


Surface

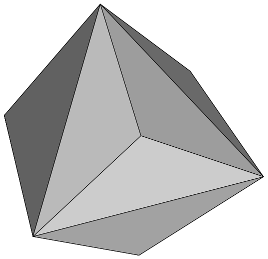


Polygon Mesh

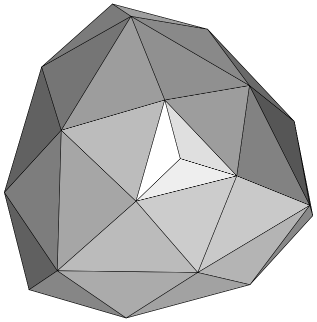
Subdivision Surfaces



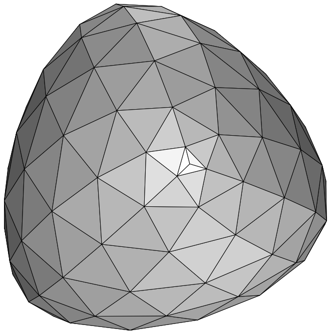
Control Mesh



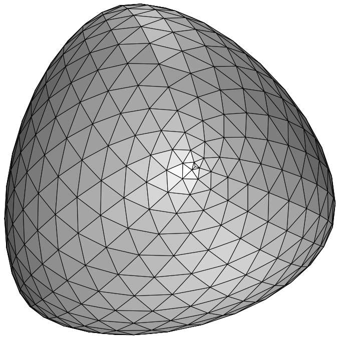
Refined Mesh
After One Iteration



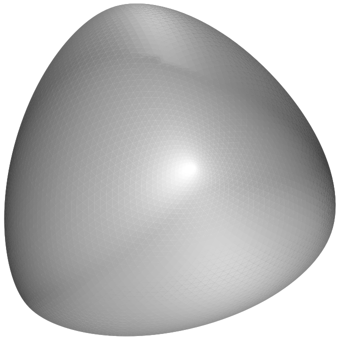
Refined Mesh
After Two Iterations



Refined Mesh
After Three Iterations



Refined Mesh
After Four Iterations



Limit Surface

Subdivision Wavelets

- subdivision wavelets are implicitly associated with subdivision surfaces
- subdivision wavelets provide multiresolution representations of polygon meshes

Applications of Subdivision Surfaces/Wavelets

- computer graphics, rendering
- animation
- volume morphing
- gaming
- biomedical computing
- computer-aided design and manufacturing
- geometric modelling
- scientific visualization
- finite element analysis, computational fluid dynamics

Geri’s Game: Subdivision Surfaces

- 1998 Academy Award Winner, Short Film (Animated)
- from Pixar Animation Studios
- goal to take human and cloth animation to new heights
- one of first animated films to make use of subdivision surfaces
- subdivision surfaces used to model skin of Geri’s head, his hands, and his clothing, including his jacket, pants, shirt, tie, and shoes

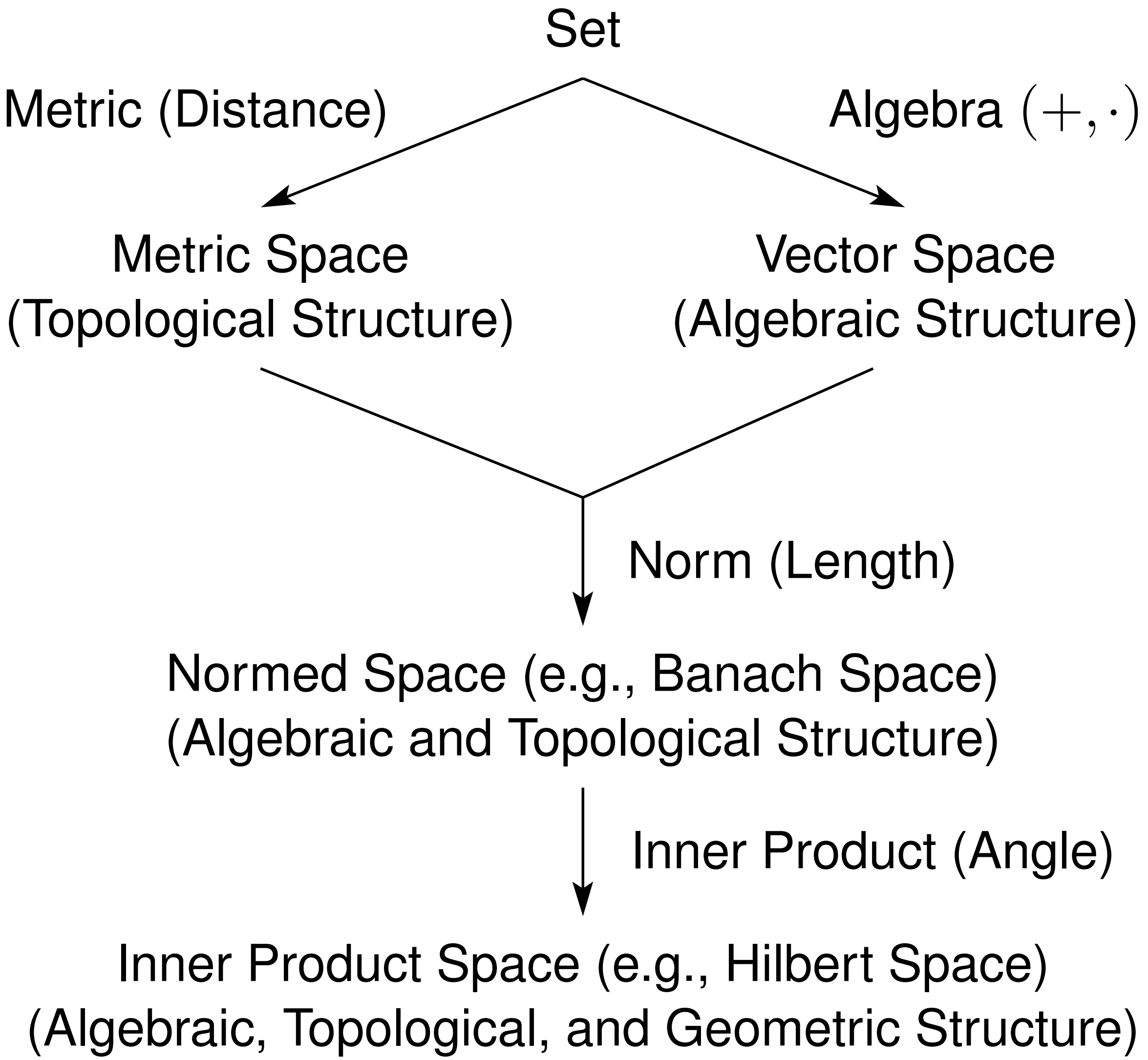
Part 2

Mathematical Preliminaries

Objective

- The goal is to introduce mathematical background (e.g., definitions, terminology, and concepts) that are helpful in better understanding wavelets and multirate signal processing.
- In particular, we primarily seek to:
    1. gain a better understanding of the mathematical structure underlying Hilbert spaces;
    2. consider how some topics from Fourier analysis relate to Hilbert spaces.
- The coursepack covers all of the topics from this presentation in considerable depth.
- To reduce the risk of math-induced comas in the student populace, we will not cover these topics in as much detail as in the coursepack.

From Sets to Spaces



Set

Metric (Distance)

Algebra $(+,\cdot)$

Metric Space
(Topological Structure)

Vector Space
(Algebraic Structure)

Norm (Length)

Normed Space (e.g., Banach Space)
(Algebraic and Topological Structure)

Inner Product (Angle)

Inner Product Space (e.g., Hilbert Space)
(Algebraic, Topological, and Geometric Structure)

Section 2.1

Sets

Sets

- A **set** is an unordered collection of (distinct) objects called **elements**.
- To denote that an object x is an element of the set A, we write $x \in A$.
  Example. $1 \in \{1,2,3\}$.
- The set containing no elements is called the **empty set** and denoted $\emptyset$.
- Two sets are **equal** if they contain exactly the same elements.
  Example. $\{1,2,3\} = \{3,1,2\} = \{2,1,3\} = \{3,2,1\}$.
- A set B is said to be a **subset** of a set A, denoted $B \subset A$, if every element of B is an element of A.
  Example. $\{1,2\} \subset \{1,2,3\}$ and $\{1,2,3\} \subset \{1,2,3\}$.
- If $B \subset A$ and $B \neq A$, then B is said to be a **proper subset** of A. A subset that is not proper is called **improper**.
  Example. $\{1,3\}$ is a proper subset of $\{1,3,5\}$, while $\{1,2\}$ is an improper subset of $\{1,2\}$.
- Two sets A and B are said to be **disjoint** if they have no common elements.
  Example. $\{0,2,4\}$ and $\{1,3,5\}$ are disjoint.

Commonly-Used Sets

- Some commonly-used sets include:
  - the **natural numbers** $\mathbb{N} = \{1, 2, 3, \ldots\}$ (i.e., positive integers);
  - the **nonnegative integers** $\mathbb{Z}^* = \{0, 1, 2, \ldots\}$;
  - the **integers** $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$;
  - the **rational numbers** $\mathbb{Q} = \{x : x = p/q,$ where $p, q \in \mathbb{Z}, q \neq 0\}$;
  - the **real numbers** $\mathbb{R}$; and
  - the **complex numbers** $\mathbb{C} = \{z : z = x + jy$ for $x, y \in \mathbb{R}$ and $j^2 = -1\}$.
- The sets $\mathbb{N}$, $\mathbb{Z}^*$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ ***do not include infinity***. So, for example, while there are infinitely many integers in $\mathbb{Z}$, every one of these integers is (by definition) finite.
- Since sets consisting of ***intervals on*** $\mathbb{R}$ are often used, we have the following notation for concisely specifying such sets. For $a, b \in \mathbb{R}$:

$$\begin{aligned}
&[a,b] = \{x \in \mathbb{R} : a \leq x \text{ and } x \leq b\}, && (a,b) = \{x \in \mathbb{R} : a < x \text{ and } x < b\},\\
&[a,b) = \{x \in \mathbb{R} : a \leq x \text{ and } x < b\}, && (a,b] = \{x \in \mathbb{R} : a < x \text{ and } x \leq b\},\\
&[a,\infty) = \{x \in \mathbb{R} : x \geq a\}, && (-\infty,b] = \{x \in \mathbb{R} : x \leq b\},\\
&(a,\infty) = \{x \in \mathbb{R} : x > a\}, \quad \text{and} && (-\infty,b) = \{x \in \mathbb{R} : x < b\}.
\end{aligned}$$

Sequences

- A **sequence** is a collection of elements from a (nonempty) set X indexed by a (nonempty) set I, and is denoted as $(x_n)_{n \in I}$ or simply (x_n), where $x_n \in X$ and $n \in I$. The set I is referred to as an **index set**.
- A sequence comprised of n elements, where n is finite, is called an **ordered n-tuple**.
- **Example.** The sequence $(x_n)_{n \in \mathbb{Z}}$ of real numbers given by $x_n = n + \frac{1}{2}$ is $(\ldots, -\frac{3}{2}, -\frac{1}{2}, \frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \ldots)$ which corresponds to the following mapping:

Index	...	-2	-1	0	1	2	...
Element	...	$-\frac{3}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	$\frac{3}{2}$	$\frac{5}{2}$	...

- In (one-dimensional) digital signal processing, we usually work with sequences where the index set is a subset of $\mathbb{Z}$.
- Often, instead of writing $(x_n)_{n \in I}$, we write $x[n]$, $n \in I$.
- A sequence can be easily converted to a set and vice versa.

Basic Set Operations

- The **union** of two sets A and B, denoted $A \cup B$, is the set obtained by combining the elements of A and B.
  Example. $\{1,2,3\} \cup \{3,4,5\} = \{1,2,3,4,5\}$.
- The **intersection** of two sets A and B, denoted $A \cap B$, is the set of elements common to both A and B.
  Example. $\{1,2,3\} \cap \{3,4,5\} = \{3\}$.
- The **difference** of two sets A and B, denoted $A \setminus B$ (or $A - B$), is the set consisting of all elements in A that are not in B.
  Example. $\{1,2,3\} \setminus \{3,4,5\} = \{1,2\}$.
- The **complement** of the set A in B is defined as $B \setminus A$.
- The **Cartesian product** of the sets $X_1 \times X_2 \times \ldots \times X_n$, denoted $X_1 \times X_2 \times \ldots \times X_n$, is the set of all ordered n-tuples $(x_1, x_2, \ldots, x_n)$, where $x_1 \in X_1, x_2 \in X_2, \ldots, x_n \in X_n$.
  Example. $\{1,2\} \times \{3,4\} = \{(1,3),(1,4),(2,3),(2,4)\}$.
- The n-fold Cartesian product of the set S is denoted S^n.
  Example. $\{0,1\}^2 = \{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(1,1)\}$.

Number of Elements in a Set

- The number of elements in a set S is said to be the **cardinality** of the set, and is denoted $\operatorname{card} S$.
  Example. $\operatorname{card}\{1,3,5\} = 3$ and $\operatorname{card}\mathbb{Z}$ is infinite.
- A set that has a one-to-one correspondence with $\{1,2,\ldots,n\}$, where n is finite, is said to be **finite**. A set that is not finite is said to be **infinite**.
  Example. $\{1,3,5\}$ is finite and $\mathbb{Z}$ is infinite.
- A set that has a one-to-one correspondence with the natural numbers (or equivalently the integers) is said to be **countably infinite**.
  Example. $\mathbb{N}$ and $\mathbb{Z}$ are countably infinite.
- **Theorem.** $\mathbb{Q}$ is countably infinite.
- A set that is either finite or countably infinite is said to be **countable**.
  Example. $\{1,2,3\}$, $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{Q}$ are countable.
- A set that is not countable is said to be **uncountable**.
  Example. $\mathbb{R}$ and $\mathbb{C}$ are uncountable.

Indicator Function

- The **indicator function** of a subset B of a set A, denoted χ_B, is defined as

$$\chi_B(t) = \begin{cases} 1 & \text{if } t \in B \\ 0 & \text{if } t \in A \setminus B. \end{cases}$$

Example.

The unit-step function (defined on $\mathbb{R}$) is $\chi_{[0,\infty)}$.
The unit-rectangular pulse (defined on $\mathbb{R}$) is $\chi_{[-1/2,1/2]}$.
The unit-step sequence (defined on $\mathbb{Z}$) is $\chi_{\mathbb{Z}^*}$.
The unit-impulse sequence (defined on $\mathbb{Z}$) is $\chi_{\{0\}}$.

Lebesgue Measure of a Set

- The **Lebesgue measure** assigns a size (akin to length/hypervolume) to a subset of a Euclidean space (such as $\mathbb{R}$).
- The Lebesgue measure of the set S is denoted $\mu(S)$.
- **Nonnegativity.** For any set S, $\mu(S) \geq 0$.
- **Empty set.** $\mu(\emptyset) = 0$.
- **Countable additivity.** For any countable collection $\{S_k\}_{k \in I}$ of pairwise disjoint sets, $\mu(\cup_{k \in I} S_k) = \sum_{k \in I} \mu(S_k)$.
- A set S for which $\mu(S) = 0$ is called a **null set**.
- **Countable subsets of $\mathbb{R}$.** Any countable subset S of $\mathbb{R}$ has $\mu(S) = 0$.
- **Open/closed intervals on $\mathbb{R}$.** For $a, b \in \mathbb{R}$, $\mu([a,b]) = \mu((a,b)) = \mu([a,b)) = \mu((a,b]) = b - a$.
- **Example.** $\mu(\{1,2,3\}) = 0$, $\mu(\mathbb{N}) = 0$, $\mu(\mathbb{Z}) = 0$, and $\mu(\mathbb{Q}) = 0$. $\mu([0,1] \cup [5,6) \cup (7,8)) = \mu([0,1]) + \mu([5,6)) + \mu((7,8)) = 3$.
- The term "**almost everywhere**" (often abbreviated "a.e.") means "everywhere except possibly for a null set".

  Example. $\chi_{\mathbb{Z}}$ is zero almost everywhere; $\sin$ is zero almost nowhere.

Bounds on Subsets of $\mathbb{R}$

- A set $X \subset \mathbb{R}$ is said to be **bounded from above** if there exists an **upper bound** $u \in \mathbb{R}$ such that $x \leq u$ for all $x \in X$.
  Example. $[0,1]$ is bounded from above and has upper bounds including 1, π, and 10^{1000}.
- A set $X \subset \mathbb{R}$ is said to be **bounded from below** if there exists a **lower bound** $l \in \mathbb{R}$ such that $x \geq l$ for all $x \in X$.
  Example. $[0,1]$ is bounded from below and has lower bounds including -10^{1000}, $-\pi$, and 0.
- A set that is both bounded from above and below is said to be **bounded**.
  Example. $[0,1]$ and $(0,1)$ are bounded.
- The **maximum** of $X \subset \mathbb{R}$, denoted $\max X$, is the ***element of X*** that is also an upper bound of X.
  Example. $\max[0,1] = 1$ and $\max(0,1)$ is not defined.
- The **minimum** of $X \subset \mathbb{R}$, denoted $\min X$, is the ***element of X*** that is also a lower bound of X.
  Example. $\min[0,1] = 0$ and $\min(0,1)$ is not defined.

Bounds on Subsets of $\mathbb{R}$ (Continued)

- A bounded set need not have a maximum or minimum.
- The **supremum** of $X \subset \mathbb{R}$, denoted $\sup X$, is the ***least upper bound*** of X.
  If X is not bounded from above, we define $\sup X = \infty$.
  If $X = \emptyset$, we define $\sup X = -\infty$.
  Example. $\sup[0,1] = \sup(0,1) = 1$ and $\sup \mathbb{R} = \infty$.
- The **infimum** of $X \subset \mathbb{R}$, denoted $\inf X$, is the ***greatest lower bound*** of X.
  If X is not bounded from below, we define $\inf X = -\infty$.
  If $X = \emptyset$, we define $\inf X = \infty$.
  Example. $\inf[0,1] = \inf(0,1) = 0$ and $\inf \mathbb{R} = -\infty$.

Notation: Functions Versus Function Values

- Strictly speaking, an expression like “$f(t)$” means the ***value*** of the function/sequence f at the point t.
- Unfortunately, engineers often use an expression like “$f(t)$” to refer to the ***function*** f (rather than the value of f at t), and this sloppy notation can lead to problems (e.g., ambiguity) in many situations.
- We will be more careful to distinguish between a function and its value.
- Some examples of the use of a more precise notation are given below. The symbol “$\cdot$” is sometimes used as a placeholder.

Expression	Meaning
$f+g$	sum of the functions f and g
$\lvert\cdot\rvert$	absolute value function
$f(\cdot-d)$	function f translated by d
$f(\cdot-d)(t_0)$	value at t_0 of the function f translated by d
$f(a\cdot)$	function f dilated by a
$f(a\cdot)(t_0)$	value at t_0 of the function f dilated by a
$[f(\cdot-t)]*[g(\cdot-t)]$	convolution of: f translated by t; and g translated by t
$(f*g)(\cdot-t)$	f convolved with g, followed by translation by t

Commonly-Used Sets of Functions and Sequences

- ***$L^p(I)$*****.** For $p \in [1,\infty) \cup \{\infty\}$, the set $L^p(I)$ is comprised of all (measurable) complex (or real) functions x defined on I for which

$$\begin{cases} \int_I |x(t)|^p \, dt < \infty & \text{for } p \in [1,\infty) \\ \operatorname{ess\,sup}_{t \in I} |x(t)| < \infty & \text{for } p = \infty. \end{cases}$$

- **Example.** The set $L^2(\mathbb{R})$ consists of all square-integrable (i.e., finite-energy) complex functions defined on $\mathbb{R}$.

- ***$l^p(I)$*****.** For $p \in [1,\infty) \cup \{\infty\}$, the set $l^p(I)$ is comprised of all complex (or real) sequences $\{x_k\}_{k \in I}$ defined on I for which

$$\begin{cases} \sum_{k \in I} |x_k|^p < \infty & \text{for } p \in [1,\infty) \\ \sup_{k \in I} |x_k| < \infty & \text{for } p = \infty. \end{cases}$$

- **Example.** The set $l^2(\mathbb{Z})$ consists of all square-summable (i.e., finite-energy) complex sequences defined on $\mathbb{Z}$.

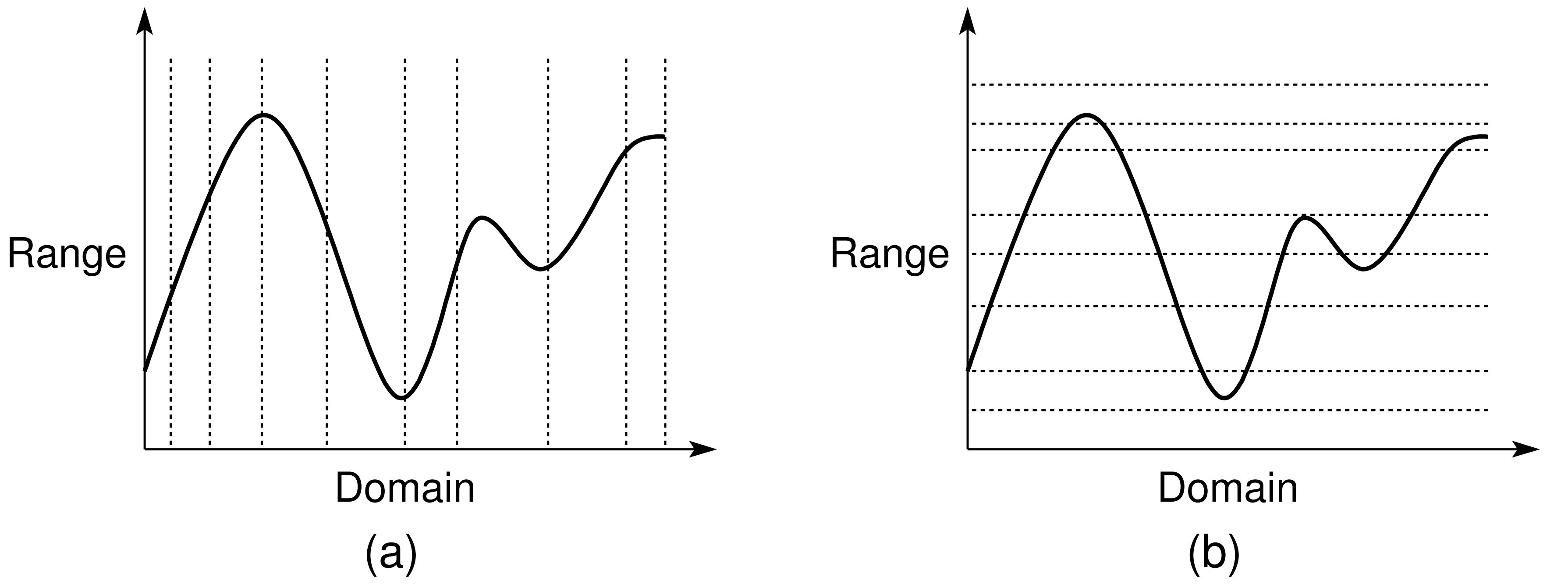
Section 2.2

Integration

Integration

- The simplest definition of integration is Riemann integration, as introduced in any first-year undergraduate calculus course.
- ***Problem:*** The Riemann integral does not exist for many reasonably well-behaved functions, and this can lead to many difficulties.
  Example. The Dirichlet function $\chi_{\mathbb{Q}}$ is bounded and zero almost everywhere; yet, $\int_a^b \chi_{\mathbb{Q}}(t)dt$ does not exist (where $a, b \in \mathbb{R}$).
- ***Solution:*** We define a new type of integration called **Lebesgue integration**.
- The Lebesgue integral is defined for many functions for which the Riemann integral fails to exist.
- Unless otherwise noted, all integrals should be interpreted in the Lebesgue sense.

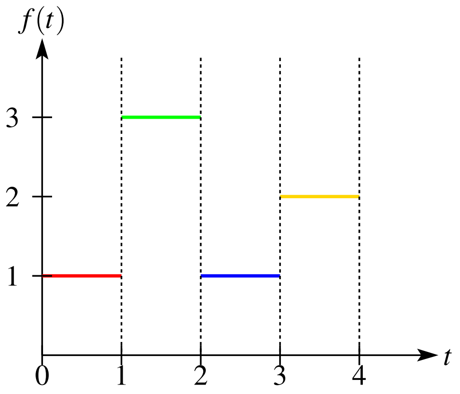
Domain Partitioning Versus Range Partitioning



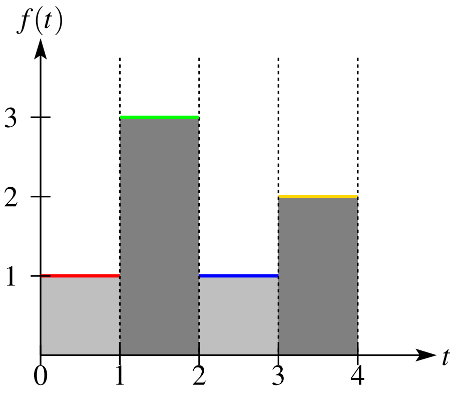
Two different approaches to integration. Partitioning of the (a) domain and (b) range of a function for integration.

- Riemann integration partitions the ***domain*** of a function.
- Lebesgue integration partitions the ***range*** of a function.

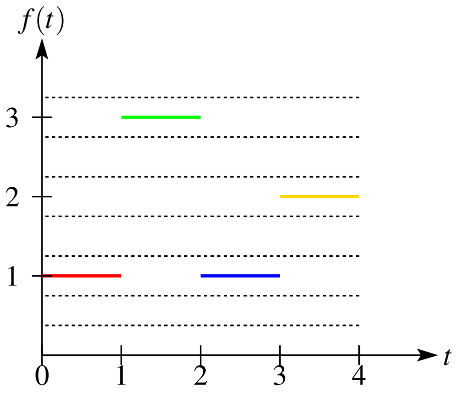
Riemann Versus Lebesgue Integration



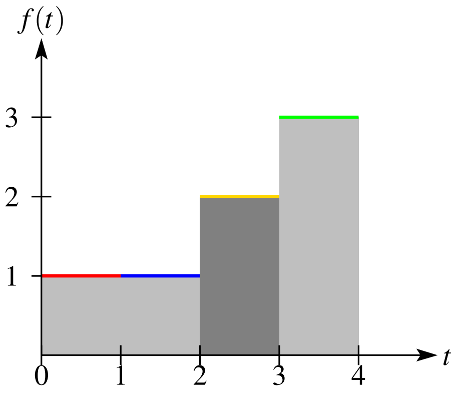
Domain Partitioning



Riemann Integral



Range Partitioning



Lebesgue Integral

Riemann integration:

$$
\begin{aligned}
&\int_0^4 f(t)dt \\
&= (1-0)(1) + (2-1)(3) + \\
&\quad (3-2)(1) + (4-3)2 \\
&= 1+3+1+2 \\
&= 7
\end{aligned}
$$

Lebesgue integration:

$$
\begin{aligned}
&\int_0^4 f(t)dt \\
&= \mu(\{t : f(t) = 1\})(1) + \mu(\{t : f(t) = 2\})(2) + \\
&\quad \mu(\{t : f(t) = 3\})(3) \\
&= (2)(1) + (1)(2) + (1)(3) \\
&= 2+2+3 \\
&= 7
\end{aligned}
$$

Lebesgue Integration

- A formal definition of the Lebesgue integral is beyond the scope of this course.
- The properties of the Lebesgue integral are very similar to the properties of the Riemann integral. E.g.,
  $\int af(t)dt = a\int f(t)dt$;
  $\int [f(t)+g(t)]dt = \int f(t)dt + \int g(t)dt$;
  $|\int f(t)dt| \leq \int |f(t)|dt$; and
  if $f > 0$, then $\int f(t)dt \geq 0$.
- If a function is Riemann integrable, it is also Lebesgue integrable and both integrals are equal.
- If f is zero almost everywhere, then the Lebesgue integral of f is zero.
  Example. $\int_a^b \chi_{\mathbb{Q}}(t)dt = 0$, where $a, b \in \mathbb{R}$.

Section 2.3

Metric Spaces

Metric Spaces

- A **metric** d on a set X is a real function defined on $X \times X$ that satisfies the following conditions:
  1. $d(x,y) \geq 0$ for all $x,y \in X$ (**nonnegativity**);
  2. $d(x,y) = 0$ if and only if $x = y$ (**strict positivity**);
  3. $d(x,y) = d(y,x)$ for all $x,y \in X$ (**symmetry**); and
  4. $d(x,y) \leq d(x,z) + d(z,y)$ for all $x,y,z \in X$ (**triangle inequality**).
- A metric is a measure of distance.
- **Example.** For $\mathbb{R}$, the usual metric is $d(x,y) = |x-y|$.
- A **metric space** is a set X with a metric d, and is denoted (X,d) or simply X when d is clear from the context.

  Example. In each of the following cases, (X,d) is a metric space:

  $X = \mathbb{R}$ and $d(x,y) = |x-y|$;

  $X = \mathbb{R}^2$ and $d[(x_1,x_2),(y_1,y_2)] = \sqrt{(x_1-y_1)^2 + (x_2-y_2)^2}$; and

  $X = L^2(\mathbb{R})$ and $d(x,y) = [\int_{\mathbb{R}} |x(t)-y(t)|^2 dt]^{1/2}$.

  $X = l^2(\mathbb{Z})$ and $d(x,y) = [\sum_{k\in\mathbb{Z}} |x_k - y_k|^2]^{1/2}$.
- A metric space has ***topological structure***.

Basic Topological Concepts

- In the metric space (X,d), an **open ball** with center x_0 and radius r (where $r \in \mathbb{R}, r > 0$) is the set $\{x \in X : d(x,x_0) < r\}$.

  Example.

  In $\mathbb{R}$, an open ball with center 0 and radius 1 is $(-1,1)$.

  In $\mathbb{R}^2$, an open ball with center $(0,0)$ and radius 1 is $\{(x_1,x_2) \in \mathbb{R}^2 : \sqrt{x_1^2+x_2^2} < 1\}$.

- An open ball with center x_0 is said to be a **neighbourhood** of x_0.

  Example. In $\mathbb{R}$, $(-\frac{1}{2},\frac{1}{2})$ and $(-1,1)$ are neighbourhoods of 0.

- Given a subset S of a metric space X, a point $x \in X$ is said to be a **boundary point** of S if every neighbourhood of x contains both elements in S and elements not in S.

  Example. A boundary point of $[0,1)$ is 0. $\mathbb{R}$ has no boundary points.

- The **boundary** of a set S, denoted $\operatorname{bdy} S$, is the set of all boundary points of S.

  Example. $\operatorname{bdy}[0,1) = \{0,1\}$. $\operatorname{bdy}\mathbb{R} = \emptyset$.

Open and Closed Sets

- A subset S of a metric space X is said to be **open** if S does not contain any of its boundary points.
  Example. $(0,1)$ is open; $[0,1)$ is not open; and $\emptyset$ is open.
- A subset S of a metric space X is said to be **closed** if S includes all of its boundary points.
  Example. $[0,1]$ is closed; $[0,1)$ is not closed; and $\emptyset$ is closed.
- The open and closed properties are not mutually exclusive.
  Example. $\emptyset$ and $\mathbb{R}$ are both open and closed.
- The **closure** of S (in X), denoted $\operatorname{clos} S$, is defined as $\operatorname{clos} S = S \cup \operatorname{bdy} S$.
  Example. $\operatorname{clos}(0,1) = [0,1]$. $\operatorname{clos} \mathbb{R} = \mathbb{R}$.
- A subset S of a metric space X is said to be **dense** in X if $\operatorname{clos} S = X$.
  Example. $\mathbb{Q}$ is dense in $\mathbb{R}$; and $L^1(\mathbb{R}) \cap L^2(\mathbb{R})$ is dense in $L^2(\mathbb{R})$.
- If S is dense in X, then every $x \in X$ can be approximated arbitrarily closely (in terms of the metric) by elements from S.

Cauchy and Convergent Sequences

- A sequence $(x_n)_{n\in\mathbb{N}}$ in a metric space (X,d) is said to be **Cauchy** if for each real number $\varepsilon > 0$ there exists $N \in \mathbb{N}$ such that $d(x_m,x_n) < \varepsilon$ for any choice of $m,n > N$. (A Cauchy sequence can be thought of as a sequence that is "***trying to converge***".)

$$x_1 \quad x_2 \quad \cdots \quad x_N \quad \underbrace{x_{N+1} \quad x_{N+2} \quad x_{N+3} \quad \cdots}_{\substack{\textbf{\textit{any two}}\text{ elements are such that} \\ \text{their distance is less than } \varepsilon}}$$

- A sequence $(x_n)_{n\in\mathbb{N}}$ in a metric space (X,d) is said to be **convergent** if there is a point x ***in*** (X,d) with the property that for each real number $\varepsilon > 0$ there exists $N \in \mathbb{N}$ such that $d(x_n,x) < \varepsilon$ whenever $n > N$. The point x is called the **limit** of the sequence $(x_n)_{n\in\mathbb{N}}$, which we denote as $\lim_{n\to\infty} x_n = x$. A sequence that is not convergent is said to be **divergent**.

$$x_1 \quad x_2 \quad \cdots \quad x_N \quad \underbrace{x_{N+1} \quad x_{N+2} \quad x_{N+3} \quad \cdots}_{\substack{\text{every element is such that} \\ \text{its distance to } x \text{ is less than } \varepsilon}}$$

Convergent and Cauchy Sequences (Continued)

- **Example.** The real sequence $(1, \frac{1}{2}, \frac{1}{4}, \ldots, \frac{1}{2^n}, \ldots)$, in the metric space $\mathbb{R}$, is Cauchy and also convergent (in $\mathbb{R}$) with limit 0.
- **Theorem.** Every convergent sequence is Cauchy.
- A Cauchy sequence need not be convergent.
  Example. The sequence $(3, 3.1, 3.14, 3.141, 3.1415, \ldots)$ which provides progressively better rational-number approximations of π is Cauchy. The sequence is not convergent in $\mathbb{Q}$ (since $\pi \notin \mathbb{Q}$), but it is convergent in $\mathbb{R}$.
- A metric space X is said to be **complete** if every Cauchy sequence in X is a convergent sequence in X.
  Example. The metric spaces $\mathbb{Z}$, $\mathbb{R}$, $\mathbb{C}$, and $[a,b]$ (where $a,b \in \mathbb{R}$) are complete, while $\mathbb{Q}$ and (a,b) are not complete.
- In a complete metric space, the Cauchy and convergent properties are equivalent.
- In practice, complete metric spaces are almost always used, since incomplete spaces can exhibit very nasty behavior.

Section 2.4

Vector Spaces

Vector Spaces

- A **vector space** over a ***scalar field*** F (such as $\mathbb{R}$ or $\mathbb{C}$) is a nonempty set V, together with two ***algebraic operations***,
  1. a mapping $(x,y) \mapsto x+y$ from $V \times V$ into V called **vector addition** and
  2. a mapping $(a,x) \mapsto ax$ from $F \times V$ into V called **scalar multiplication**,

  which satisfy the axioms of a vector space. Such a vector space is denoted $(V,F,+,\cdot)$ or simply V when the other parameters are clear from the context.
- A vector space has ***algebraic structure***.
- A vector space over the field $\mathbb{R}$ is called a **real vector space**.
- A vector space over the field $\mathbb{C}$ is called a **complex vector space**.

Axioms of a Vector Space

1. for all $x, y \in V$, $x + y \in V$ (**closure under vector addition**);
2. for all $x \in V$ and all $a \in F$, $ax \in V$ (**closure under scalar multiplication**);
3. for all $x, y \in V$, $x + y = y + x$ (**commutativity of vector addition**);
4. for all $x, y, z \in V$, $(x + y) + z = x + (y + z)$ (**associativity of vector addition**);
5. for all $x \in V$ and all $a, b \in F$, $(ab)x = a(bx)$ (**associativity of scalar multiplication**);
6. for all $x \in V$ and all $a, b \in F$, $(a + b)x = ax + bx$ (**distributivity of scalar sums**);
7. for all $x, y \in V$ and all $a \in F$, $a(x + y) = ax + ay$ (**distributivity of vector sums**);
8. there exists $0 \in V$ such that $x + 0 = x$ for all $x \in V$ (**additive identity**);
9. for all $x \in V$, there exists a $(-x) \in V$ such that $x + (-x) = 0$ (**additive inverse**); and
10. for all $x \in V$, $1x = x$, where 1 denotes the multiplicative identity of the field F (**scalar multiplication identity**).

Vector Spaces

- **Example.** $\mathbb{R}^n$.
  Choose the underlying set as $V = \mathbb{R}^n$ and the field as $F = \mathbb{R}$.
  Define vector addition as:
  $(x_1, x_2, \ldots, x_n) + (y_1, y_2, \ldots, y_n) = (x_1 + y_1, x_2 + y_2, \ldots, x_n + y_n)$.
  Define scalar multiplication as: $ax = (ax_1, ax_2, \ldots, ax_n)$.
- **Example.** $L^2(\mathbb{R})$.
  Choose the underlying set as $V = L^2(\mathbb{R})$ and the field as $F = \mathbb{C}$.
  Define vector addition as: $(x+y)(t) = x(t) + y(t)$.
  Define scalar multiplication as: $(ax)(t) = ax(t)$.
- **Example.** $l^2(\mathbb{Z})$.
  Choose the underlying set as $V = l^2(\mathbb{Z})$ and the field as $F = \mathbb{C}$.
  Define vector addition as: $(\ldots, x_{-1}, x_0, x_1, \ldots) + (\ldots, y_{-1}, y_0, y_1, \ldots) =$
  $(\ldots, x_{-1} + y_{-1}, x_0 + y_0, x_1 + y_1, \ldots)$.
  Define scalar multiplication as:
  $a(\ldots, x_{-1}, x_0, x_1, \ldots) = (\ldots, ax_{-1}, ax_0, ax_1, \ldots)$.

Vector Subspaces

- A subset of a vector space V that is itself a vector space is called a **vector subspace** of V.
  Example. The xy-plane is a vector subspace of $\mathbb{R}^3$.
- A subspace S of the vector space V is said to be **proper** if $S \neq V$ and **improper** if $S = V$.
- **Theorem.** A nonempty subset S of a vector space V over F is a vector subspace if the following conditions hold:
  1. $x + y \in S$ for all $x, y \in S$ (***closure under vector addition***); and
  2. $ax \in S$ for all $x \in S$ and all $a \in F$ (***closure under scalar multiplication***).
- Two vector subspaces V and W of the same dimensionality are said to be **disjoint** if $V \cap W = \{0\}$ (i.e., the only common vector between V and W is the zero vector).
  Example. Let W_e and W_o denote the subspaces of $L^2(\mathbb{R})$ consisting of all even and all odd functions, respectively. Then, W_e and W_o are disjoint (since only the zero function is both even and odd).

Linear Transformations

- A transformation T of a vector space V into a vector space W, where V and W have the same scalar field F, is said to be a **linear transformation** if
  1. for all $x \in V$ and all $a \in F$, $T(ax) = aT(x)$ **(homogeneity)**; and
  2. for all $x, y \in V$, $T(x+y) = T(x) + T(y)$ **(additivity)**.

  Example. Some linear transformations include: scaling, rotation, shear, and reflection in $\mathbb{R}^n$, and the Fourier transform in $L^2(\mathbb{R})$ and $l^2(\mathbb{Z})$.
- The **null space** of a linear transformation $T : V \to W$, denoted $N(T)$, is the subset of V given by $N(T) = \{x \in V : Tx = 0\}$ (i.e., the set of all vectors mapped to the zero vector under the transformation T).
- The **range space** of a linear transformation $T : V \to W$, denoted $R(T)$, is defined as $R(T) = \{y = Tx : x \in V\}$ (i.e., the set of vectors produced by applying T to each of the elements of V).
- A linear transformation P of a vector space V into itself is said to be a **projection** if $P^2 = P$ (i.e., P is **idempotent**).

  Example. In $\mathbb{R}^2$, the transformation that maps (x_1, x_2) to $(x_1, 0)$ is a projection (i.e., a projection onto the x axis).

Linear Combinations

- A **linear combination** of the vectors $v_1, v_2, \ldots, v_n$, where n is ***finite***, is an expression of the form $a_1x_1 + a_2x_2 + \ldots + a_nx_n$, where $a_1, a_2, \ldots, a_n$ are scalars.

  Example. In $\mathbb{R}^2$, $y = (1,2)$ is a linear combination of $x_1 = (1,0)$ and $x_2 = (0,1)$, namely, $y = 1x_1 + 2x_2$.
- In a vector space, the sum of an infinite number of elements is not defined, since such a sum implicitly requires the notion of a limit, which is lacking in a vector space.
- For any nonempty subset A of a vector space V, the set of all (finite) linear combinations of vectors in A is called the **span** of A, denoted $\operatorname{span} A$.

  Example.

  In $\mathbb{R}^3$, $\operatorname{span}\{(1,0,0),(0,1,0)\}$ is the xy plane.

  In the vector space of polynomials of order 3 or less, $\operatorname{span}\{1,t\}$ is the set of all constant and linear functions.

Linear Independence

- If x and y are nonzero vectors and $x = ay$ for some scalar a, then x and y are said to be **collinear**.
  Example. In $\mathbb{R}^2$, $(1,1)$ and $(2,2)$ are collinear.
- A (nonempty) finite sequence $(x_n)_{n \in I}$ of vectors in a vector space V is said to be **linearly independent** if the only sequence $(a_n)_{n \in I}$ of scalars satisfying $\sum_{n \in I} a_n x_n = 0$ is the trivial solution with $a_n = 0$ for all $n \in I$. An infinite sequence A of vectors in V is said to be linearly independent if every (nonempty) finite subsequence of A is linearly independent. A sequence that is not linearly independent is called **linearly dependent**.
- **Example.**
  In $\mathbb{R}^3$, $\{(1,0,0),(0,1,0)\}$ is linearly independent.
  In $l^2(\mathbb{Z})$, $(\delta[\cdot - n])_{n \in \mathbb{Z}}$ is linearly independent.
  In $\mathbb{R}^2$, $\{(1,2),(2,4)\}$ is linearly dependent.

Hamel Bases and Dimensionality

- A sequence A of vectors in a vector space V is said to be a **Hamel basis** of V if A is linearly independent and $\operatorname{span} A = V$.
  Example. $\{(1,0,0),(0,1,0),(0,0,1)\}$ is a Hamel basis for $\mathbb{R}^3$.
- The cardinality of any Hamel basis of a vector space V is said to be the **dimension** of V, denoted $\dim V$.
  Example.
  A Hamel basis for $V = \mathbb{R}^2$ is $\{(1,0),(0,1)\}$; hence, $\dim V = 2$.
  Any Hamel basis for $V = L^2(\mathbb{R})$ (i.e., finite-energy functions defined on $\mathbb{R}$) is uncountable; hence, $\dim V$ is infinite.
- If the dimension of V is finite, we say that V is **finite dimensional**. Otherwise, we say that V is **infinite dimensional**.
- Often, in signal processing, we must deal with infinite-dimensional spaces (e.g., function and sequence spaces).

Inner Sums and Algebraic Complements

- If V and W are subspaces of the vector space U, then the **inner sum** of V and W, denoted $V+W$, is the space consisting of all points $x=v+w$ where $v \in V$ and $w \in W$. ***[Note: $V+W$ is not the same as $\neq V \cup W$.]***
  Example. Let U and V be subspaces of $\mathbb{R}^2$, where $U = \operatorname{span}\{(1,0)\}$ and $V = \operatorname{span}\{(0,1)\}$. Then, $U+V = \mathbb{R}^2$.
- Let V and W be subspaces of the vector space U. If $U = V+W$ and V and W are disjoint, W is the called the **algebraic complement** of V in U. (Similarly, V is the algebraic complement of W in U.)
  Example. Let W_e and W_o be the subspaces of $V = L^2(\mathbb{R})$ consisting of even and odd functions, respectively. Since any function can be expressed as the sum of an even and odd function, we have $W_e + W_o = L^2(\mathbb{R})$. Since W_e and W_o are disjoint, W_e and W_o are algebraic complements.
- **Theorem.** The algebraic complement always exists.
- **Theorem.** Let V and W be subspaces of a vector space U. Then for each $x \in V+W$, there is a unique $v \in V$ and a unique $w \in W$ such that $x = v+w$ if and only if V and W are disjoint (i.e., a vector has a unique decomposition in terms of algebraic complements).

Direct Sums

- Let V and W be subspaces of the vector space U. If U and V are disjoint, the (isomorphic form of the) **direct sum** of U and V, denoted $U \oplus W$, is $U + W$.

  Example. Let W_e and W_o be the subspaces of $L^2(\mathbb{R})$ consisting of all even and all odd functions, respectively. Since any function can be expressed as the sum of an even and odd function, we have $W_e + W_o = L^2(\mathbb{R})$. Since W_e and W_o are disjoint, we may also write $W_e \oplus W_o = L^2(\mathbb{R})$.

Section 2.5

Normed Spaces

Norms

- A norm on a vector space V over the field F is a mapping $\|\cdot\|$ of V into $\mathbb{R}$ with the following properties:
  1. for all $x \in V$, $\|x\| \geq 0$ (**nonnegativity**);
  2. $\|x\| = 0$ if and only if $x = 0$ (**strict positivity**);
  3. for all $x \in V$ and all $a \in F$, $\|ax\| = |a|\,\|x\|$ (**homogeneity**); and
  4. for all $x, y \in V$, $\|x+y\| \leq \|x\| + \|y\|$ (**triangle inequality**).
- A norm is a measure of length.
- A norm is continuous (which means that the order of limits and norms can be interchanged).
- **Example.**
  For $\mathbb{R}$, the function $\|x\| = |x|$ is a norm.
  For $L^2(\mathbb{R})$, the function $\|x\| = [\int_{\mathbb{R}} |x(t)|^2\,dt]^{1/2}$ is a norm.
  For $l^2(\mathbb{Z})$, the function $\|(\ldots, x_{-1}, x_0, x_1, \ldots)\| = [\sum_{n\in\mathbb{Z}} |x_n|^2]^{1/2}$ is a norm.
- A norm induces a metric.
  Given a norm $\|\cdot\|$, the function $d(x,y) = \|x-y\|$ is a metric.
  Example. For $\mathbb{R}$, the function $\|x\| = |x|$ is a norm, which induces the metric $d(x,y) = \|x-y\| = |x-y|$.

Normed Spaces

- A vector space V with a norm $\|\cdot\|$ defined on V is called a **normed space**, and is denoted $(V, \|\cdot\|)$ or simply V when the norm is implied from the context.
- A normed space has a metric induced by the norm. Thus, a normed space is also a metric space.
- A normed space has ***algebraic and topological structure***.
- **Example.** $\mathbb{C}$. The set $\mathbb{C}$ with the norm $\|x\| = |x|$.
- **Example.** $L^2(\mathbb{R})$. The set $L^2(\mathbb{R})$ with the norm $\|x\| = [\int_{\mathbb{R}} |x(t)|^2\, dt]^{1/2}$.
- **Example.** $l^2(\mathbb{Z})$. The set $l^2(\mathbb{Z})$ with the norm $\|(\ldots, x_{-1}, x_0, x_1, \ldots)\| = [\sum_{n\in\mathbb{Z}} |x_n|^2]^{1/2}$.

Infinite Series

- Since normed spaces have topological structure, we can define an infinite series.
- An infinite series is defined in terms of a limit of the sequence of partial sums as follows:
  $\sum_{k\in\mathbb{N}} x_k = \lim_{n\to\infty} s_n$ where $s_n = \sum_{k=1}^{n} x_k$

ω-Independence

- A (finite or infinite) sequence $(x_n)_{n \in I}$ of vectors in a normed space is said to be **ω-independent** if whenever the series $\sum_{n \in I} a_n x_n$ is convergent and equal to zero for some sequence $(a_n)_{n \in I}$ of scalars, then necessarily $a_n = 0$ for all $n \in I$.
- The concept of ω-independence is similar to that of linear independence except that, in the former case, ***infinite*** linear combinations are also permitted.
- If I is finite, ω-independence and linear independence are essentially the same.
- **Theorem.** If a sequence of vectors is ω-independent, it is also linearly independent.

  The converse of the preceding statement is not true. In this sense, ω-independence is a stronger form of independence than linear independence.
- When working with infinite-dimensional spaces, we are almost always interested in ω-independence (and not linear independence).

Section 2.6

Inner Product Spaces

Inner Products

- An **inner product** on a vector space V over a field F is a mapping $\langle \cdot, \cdot \rangle$ of $V \times V$ into F with the following properties:
  1. $\langle x, x \rangle \geq 0$ for all $x \in V$ (**nonnegativity**);
  2. for all $x \in V$, $\langle x, x \rangle = 0$ if and only if $x = 0$ (**strict positivity**);
  3. $\langle x, y \rangle^* = \langle y, x \rangle$ for all $x, y \in V$ (**conjugate symmetry**);
  4. $\langle ax, y \rangle = a \langle x, y \rangle$ for all $x, y \in V$ and all $a \in F$ (**homogeneity**); and
  5. $\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$ for all $x, y, z \in V$ (**additivity**).
- An inner product is continuous (which means that the order of limits and inner products can be interchanged).
- An inner product $\langle \cdot, \cdot \rangle$ induces a norm. Given an inner product $\langle \cdot, \cdot \rangle$, the function $\|x\| = \langle x, x \rangle^{1/2}$ is a norm.
- The **angle** $\theta_{x,y}$ between two (nonzero) vectors x and y is defined as $\cos \theta_{x,y} = \frac{\langle x, y \rangle}{\|x\| \|y\|}$. [Note: $-1 \leq \frac{\langle x, y \rangle}{\|x\| \|y\|} \leq 1$.]
- An inner product facilitates the measure of angles. It imposes ***geometric structure*** on a set.
- **Example.**
  For $\mathbb{R}^n$, an inner product is $\langle (x_1, x_2, \ldots, x_n), (y_1, y_2, \ldots, y_n) \rangle = \sum_{k=1}^{n} x_k y_k$.
  For $\mathbb{C}^n$, an inner product is $\langle (x_1, x_2, \ldots, x_n), (y_1, y_2, \ldots, y_n) \rangle = \sum_{k=1}^{n} x_k y_k^*$.

Inner Product Spaces

- A vector space V with an inner product defined on V is called an **inner product space**, and is denoted $(V, \langle \cdot, \cdot \rangle)$ or simply V when the inner product is implied from the context.
- An inner product space also has a norm and metric induced by the inner product. Thus, an inner product space is also a normed space and a metric space.
- An inner product space has ***geometric structure*** in addition to ***algebraic and topological structure***.
- **Example.** $\mathbb{R}^n$. The vector space $\mathbb{R}^n$ with the inner product $\langle (x_1, x_2, \ldots, x_n), (y_1, y_2, \ldots, y_n) \rangle = \sum_{k=1}^{n} x_k y_k$ (i.e., the dot product).
- **Example.** $\mathbb{C}^n$. The vector space $\mathbb{C}^n$ with the inner product $\langle (x_1, x_2, \ldots, x_n), (y_1, y_2, \ldots, y_n) \rangle = \sum_{k=1}^{n} x_k y_k^*$ (i.e., the dot product).
- **Example.** $L^2(\mathbb{R})$. The vector space $L^2(\mathbb{R})$ with the inner product $\langle x, y \rangle = \int_{\mathbb{R}} x(t) y^*(t) dt$.
- **Example.** $l^2(\mathbb{Z})$. The vector space $l^2(\mathbb{Z})$ with the inner product $\langle (\ldots, x_{-1}, x_0, x_1, \ldots), (\ldots, y_{-1}, y_0, y_1, \ldots) \rangle = \sum_{k \in \mathbb{Z}} x_k y_k^*$.

Hilbert Spaces

- A **Hilbert space** is a complete inner product space (complete in the metric induced by the inner product).
- Inner product spaces lacking completeness are typically very badly behaved (e.g., an orthonormal basis may fail to exist, and so on).
- The inner product spaces used in engineering are essentially always Hilbert spaces.
- **Example.** $\mathbb{R}^n$, $\mathbb{C}^n$, $L^2(\mathbb{R})$, and $l^2(\mathbb{Z})$ are Hilbert spaces.
- **Theorem.** A closed subspace of a Hilbert space is itself a Hilbert space.

Orthogonality

- **Orthogonal vectors.** Two vectors x and y in an inner product space V are said to be **orthogonal**, denoted $x \perp y$, if $\langle x, y \rangle = 0$.
  Example. In $\mathbb{R}^2$, $v_1 = (1,0)$ and $v_2 = (0,1)$ are orthogonal, since $\langle v_1, v_2 \rangle = \langle (1,0), (0,1) \rangle = (1)(0) + (0)(1) = 0$.
- **Vector Orthogonal to Set.** For a subset A of an inner product space V and a vector $x \in V$, if $x \perp y$ for all $y \in A$, we say that x is orthogonal to A, denoted $x \perp A$.
  Example. In $\mathbb{R}^3$, $(0,0,1) \perp \{(1,0,0),(0,1,0),(1,1,0)\}$.
- **Set Orthogonal to Set.** For two subsets A and B of an inner product space V, if $x \perp y$ for all $x \in A$ and all $y \in B$, we say that A is orthogonal to B, denoted $A \perp B$.
  Example. In $\mathbb{R}^3$, $\{(1,0,0),(0,1,0)\} \perp \{(0,0,1),(0,0,2)\}$.
- **Orthogonal subspaces.** Two subspaces U and W of an inner product space are said to be orthogonal if $U \perp W$.

Orthogonality (Continued)

- **Orthogonal and orthonormal sequences.** A sequence of vectors $(x_n)_{n \in I}$ in an inner product space is said to be **orthogonal** if $x_n \perp x_m$ for all $m, n \in I$, $m \neq n$, and **orthonormal** if in addition to the preceding condition $\langle x_n, x_n \rangle = 1$ (i.e., $\|x_n\| = 1$) for all $n \in I$.
- **Example.** Consider the sequence (v_1, v_2) of vectors in $\mathbb{R}^2$, where $v_1 = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$ and $v_2 = \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$. We have:
  $\langle v_1, v_2 \rangle = \left\langle \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right), \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right) \right\rangle = \left(\frac{1}{\sqrt{2}}\right)\left(-\frac{1}{\sqrt{2}}\right) + \left(\frac{1}{\sqrt{2}}\right)\left(\frac{1}{\sqrt{2}}\right) = 0,$
  $\langle v_1, v_1 \rangle = \left\langle \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right), \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right) \right\rangle = \left(\frac{1}{\sqrt{2}}\right)\left(\frac{1}{\sqrt{2}}\right) + \left(\frac{1}{\sqrt{2}}\right)\left(\frac{1}{\sqrt{2}}\right) = 1$, and
  $\langle v_2, v_2 \rangle = \left\langle \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right), \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right) \right\rangle = \left(-\frac{1}{\sqrt{2}}\right)\left(-\frac{1}{\sqrt{2}}\right) + \left(\frac{1}{\sqrt{2}}\right)\left(\frac{1}{\sqrt{2}}\right) = 1$
  Thus, (v_1, v_2) is both orthogonal and orthonormal.
- **Theorem.** An orthonormal sequence is ω-independent.
- **Pythagorean theorem.** If two vectors x and y in an inner product space are orthogonal, then $\|x+y\|^2 = \|x\|^2 + \|y\|^2$.

Orthogonal Complements

- Let W be a nonempty subset of an inner product space V. The set of all elements of V orthogonal to W, denoted $W^\perp$, is called the **orthogonal complement** of W (in V) (i.e., $W^\perp = \{v \in V : v \perp W\}$).

  Example. Let $S \subset \mathbb{R}^3$ where $S = \{(0,0,1)\}$. Then, $S^\perp$ is the xy-plane.

Orthogonal Projections

- A projection P on an inner product space is said to be **orthogonal** if its range and null spaces are orthogonal (i.e., $R(P) \perp N(P)$).
- A projection that is not orthogonal is called **oblique**.
- **Projection theorem.** If W is a closed subspace of a Hilbert space V, then every element $x \in V$ has a unique decomposition of the form $x = y + z$ where $y \in W$ and $z \in W^\perp$.
- **Best approximation.** Let W be a closed subspace of a Hilbert space V, and let $x \in V$. Further, let P be the orthogonal projection of V onto W. There exists a ***unique*** vector $y \in W$ that is ***closest*** to x as given by $y = Px$ (closest in the sense of minimizing $\|y - x\|$).

Orthonormal Bases

- An orthonormal sequence $(e_n)_{n \in I}$ of vectors in an inner product space V is said to be an **orthonormal basis** of V if, for every $x \in V$, there exists a unique scalar sequence $(a_n)_{n \in I}$ in $l^2(I)$ such that $x = \sum_{n \in I} a_n e_n$.
- Equivalently, an orthonormal sequence E is an orthonormal basis for the inner product space V if $\operatorname{span} E$ is dense in V (i.e., $V = \operatorname{clos} \operatorname{span} E$).
- **Example.**
  An orthonormal basis of $\mathbb{R}^3$ is given by $((1,0,0),(0,1,0),(0,0,1))$.
  An orthonormal basis of $l^2(\mathbb{Z})$ is given by $(\delta[\cdot - k])_{k \in \mathbb{Z}}$.
- Unless the space in question is finite-dimensional, an orthonormal basis is not a basis in the algebraic sense. That is, an orthonormal basis is only a Hamel basis in the case of finite-dimensional spaces.
- **Existence of orthonormal basis.** Every Hilbert space has an orthonormal basis.

Orthonormal Bases (Continued)

- **Expansion coefficients.** Let $(e_n)_{n \in I}$ be an orthonormal basis of an inner product space V. Then, each $x \in V$ can be expressed as $x = \sum_{n \in I} a_n e_n$ where $a_n = \langle x, e_n \rangle$.
- **Example.** Consider the orthonormal basis (e_1, e_2) for $\mathbb{R}^2$, where $e_1 = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$ and $e_2 = (-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$. The vector $x = (2, 1)$ can be expressed as $x = a_1 e_1 + a_2 e_2$, where
$a_1 = \langle x, e_1 \rangle = (2)(\frac{1}{\sqrt{2}}) + (1)(\frac{1}{\sqrt{2}}) = \frac{3}{\sqrt{2}}$ and
$a_2 = \langle x, e_2 \rangle = (2)(-\frac{1}{\sqrt{2}}) + (1)(\frac{1}{\sqrt{2}}) = -\frac{1}{\sqrt{2}}$.
- **Parseval identity**. Let $(e_n)_{n \in I}$ be an orthonormal basis of a Hilbert space V. Then, for all $x, y \in V$, $\langle x, y \rangle = \sum_{n \in I} \langle x, e_n \rangle \langle y, e_n \rangle^*$, which, for $x = y$, simplifies to $\|x\|^2 = \sum_{n \in I} |\langle x, e_n \rangle|^2$ (i.e., an orthonormal basis preserves inner products and norms).

Orthogonal Projections (Continued)

- **Finding orthogonal projection.** Let W be a closed subspace of an Hilbert space V, and let $(e_n)_{n \in I}$ be an orthonormal basis for W. Further, let P denote the orthogonal projection of V onto W. Then, P is given by $Px = \sum_{n \in I} \langle x, e_n \rangle e_n$, where $x \in V$.
- **Example.** Let (e_1, e_2) be an orthonormal basis for a subspace W of $\mathbb{R}^3$, where $e_1 = (\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}})$ and $e_2 = (-\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}})$. Find the orthogonal projection y of $x = (1,2,1)$ onto W. We have
$$y = \langle x, e_1 \rangle e_1 + \langle x, e_2 \rangle e_2 = \left\langle (1,2,1), (\tfrac{1}{\sqrt{2}}, 0, \tfrac{1}{\sqrt{2}}) \right\rangle (\tfrac{1}{\sqrt{2}}, 0, \tfrac{1}{\sqrt{2}}) + \left\langle (1,2,1), (-\tfrac{1}{\sqrt{2}}, 0, \tfrac{1}{\sqrt{2}}) \right\rangle (-\tfrac{1}{\sqrt{2}}, 0, \tfrac{1}{\sqrt{2}}) = \sqrt{2}(\tfrac{1}{\sqrt{2}}, 0, \tfrac{1}{\sqrt{2}}) + 0 = (1,0,1).$$

Biorthogonality

- Two sequences $(x_n)_{n\in I}$ and $(y_n)_{n\in I}$ of vectors in an inner product space are said to be **biorthogonal** if $x_m \perp y_n$ for all $m,n \in I$, $m \neq n$. If, in addition, $\langle x_n, y_n\rangle = 1$ for all $n \in I$, then the sequences are said to be **biorthonormal**.
- An orthogonal sequence is biorthogonal with itself.
  An orthonormal sequence is biorthonormal with itself.
- **Example.** Let (e_1, e_2) and $(\tilde{e}_1, \tilde{e}_2)$ be sequences of vectors in $\mathbb{R}^2$, where $e_1 = (1,1)$, $e_2 = (-1,1)$, $\tilde{e}_1 = (\frac{1}{2}, \frac{1}{2})$, and $\tilde{e}_2 = (-\frac{1}{2}, \frac{1}{2})$. We have:
  $\langle e_1, \tilde{e}_1\rangle = (1)(\frac{1}{2}) + (1)(\frac{1}{2}) = 1$, $\langle e_2, \tilde{e}_2\rangle = (-1)(-\frac{1}{2}) + (1)(\frac{1}{2}) = 1$,
  $\langle e_1, \tilde{e}_2\rangle = (1)(-\frac{1}{2}) + (1)(\frac{1}{2}) = 0$, and $\langle e_2, \tilde{e}_1\rangle = (-1)(\frac{1}{2}) + (1)(\frac{1}{2}) = 0$.
  Thus, (e_1, e_2) and $(\tilde{e}_1, \tilde{e}_2)$ are both biorthogonal and biorthonormal.

Riesz Bases

- Although an orthonormal basis is often convenient to use, orthonormality can place too many constraints on the choice of basis vectors.
- In an infinite-dimensional space, ω-independence alone is not sufficient to ensure a well-behaved basis. A stronger condition is required, which leads to the notion of a Riesz basis.
- A sequence $(e_n)_{n \in I}$ of vectors in a Hilbert space V is said to be a **Riesz basis** of V if, for every $x \in V$, there exists a unique scalar sequence $(a_n)_{n \in I}$ in $l^2(I)$ such that $x = \sum_{n \in I} a_n e_n$, and there exist real numbers $A, B > 0$ (independent of x) satisfying the **Riesz condition** $A \sum_{n \in I} \left|a_n\right|^2 \leq \left\|x\right\|^2 \leq B \sum_{n \in I} \left|a_n\right|^2$ (or equivalently, $\frac{1}{B} \left\|x\right\|^2 \leq \sum_{n \in I} \left|a_n\right|^2 \leq \frac{1}{A} \left\|x\right\|^2$). The constants A and B are referred to as the lower and upper **Riesz bounds**, respectively.
- An orthonormal basis is a special case of a Riesz basis with $A = B = 1$.
- A Riesz basis is a Hamel basis only in the finite-dimensional case.
- **Theorem.** A Riesz basis is ω-independent.

Dual Riesz Bases

- Let $(e_n)_{n \in I}$ be a Riesz basis of a Hilbert space V with lower and upper Riesz bounds A and B, respectively. Then, there exists another Riesz basis $(\tilde{e}_n)_{n \in I}$ of V with lower and upper Riesz bounds $\frac{1}{B}$ and $\frac{1}{A}$, respectively, such that for all $x \in V$, $x = \sum_{n \in I} \langle x, \tilde{e}_n \rangle e_n = \sum_{n \in I} \langle x, e_n \rangle \tilde{e}_n$. We call $(\tilde{e}_n)_{n \in I}$ the **dual Riesz basis** of $(e_n)_{n \in I}$.
- **Theorem.** Dual Riesz bases are biorthonormal.
- To compute the ***expansion coefficients*** of a vector in terms of a Riesz basis, the dual basis is used as shown above.
- **Example.** Let (e_1, e_2) and $(\tilde{e}_1, \tilde{e}_2)$ be dual Riesz bases of $\mathbb{R}^2$, where $e_1 = (1,1)$, $e_2 = (-1,1)$, $\tilde{e}_1 = (\frac{1}{2}, \frac{1}{2})$, and $\tilde{e}_2 = (-\frac{1}{2}, \frac{1}{2})$. Express $x = (3,1)$ in terms of the basis (e_1, e_2). We have: $x = a_1 e_1 + a_2 e_2$, where
  $a_1 = \langle x, \tilde{e}_1 \rangle = \left\langle (3,1), (\frac{1}{2}, \frac{1}{2}) \right\rangle = 2$ and
  $a_2 = \langle x, \tilde{e}_2 \rangle = \left\langle (3,1), (-\frac{1}{2}, \frac{1}{2}) \right\rangle = -1$.

Section 2.7

Miscellany

Support of a Function

- The **support** of a function f, denoted $\operatorname{supp} f$, is the closure of the set $\{t : f(t) \neq 0\}$ (i.e., the smallest closed set that contains all of the points where f is nonzero).
  Example. $\operatorname{supp} \operatorname{rect} = [-\frac{1}{2}, \frac{1}{2}]$ and $\operatorname{supp} \operatorname{sinc} = \mathbb{R}$.
- A function f defined on $\mathbb{R}$ is said to have **compact support** if $\operatorname{supp} f \subset [a,b]$ for $a,b \in \mathbb{R}$. (Note: The terms “finite duration” and “time limited” are synonymous with compact support.)
  Example. The rect function has compact support, while the sinc function does not have compact support.

Moments

- The kth **moment** of a sequence x defined on $\mathbb{Z}$ is given by $m_k = \sum_{n \in \mathbb{Z}} n^k x[n]$ (i.e., $m_k = \langle x, (\cdot)^k \rangle$).
- The kth moment of a function x defined on $\mathbb{R}$ is given by $m_k = \int_{-\infty}^{\infty} t^k x(t) dt$ (i.e., $m_k = \langle x, (\cdot)^k \rangle$).
- Moments are essentially inner products with monomials. Since monomials/polynomials play an important role in many contexts, moments are often of interest.
- A function or sequence f is said to have p **vanishing moments** if its first p moments vanish (i.e., $m_0 = m_1 = \ldots = m_{p-1} = 0$, where m_k is the kth moment of f).

L^p Spaces

- The $L^p(I)$ spaces are a family of complete normed (i.e., Banach) spaces, where the parameter $p \in [1,\infty) \cup \{\infty\}$.
- The underlying set is comprised of all (measurable) complex (or real) ***functions*** $x(t)$ defined on I such that

$$\begin{cases} \int_I |x(t)|^p \, dt < \infty & p \in [1,\infty) \\ \operatorname{ess\,sup}_{t \in I} |x(t)| < \infty & p = \infty. \end{cases}$$

- The cases of $p = 1$, $p = 2$, and $p = \infty$ correspond to the spaces of absolutely-integrable, square-integrable, and essentially-bounded (i.e., almost-everywhere bounded) functions, respectively. The set I is often $\mathbb{R}$ or a closed interval in $\mathbb{R}$.
- Vector addition and scalar multiplication are defined in the straightforward way for functions.
- The norm is given by

$$\|x\|_p = \begin{cases} \left[\int_I |x(t)|^p \, dt\right]^{1/p} & p \in [1,\infty) \\ \operatorname{ess\,sup}_{x \in I} |x(t)| & p = \infty. \end{cases}$$

L^p Spaces (Continued)

- When $p = 2$, we obtain a Hilbert space (i.e., $L^2(I)$), where the inner product is given by

$$\langle x, y \rangle = \int_I x(t) y^*(t) dt.$$

- In the L^p spaces, equality of functions is not defined as being pointwise. Rather, two functions are equal if they differ only on set of measure zero.
- Many complicating factors arise as a result of the manner in which equality is defined (e.g., functions are not well defined at individual points)
- Technically, the elements of L^p spaces are not functions, but rather sets of functions (called equivalence classes) that differ only on a set of measure zero.
- The L^p spaces are not nested (e.g., $L^1(\mathbb{R}) \not\subset L^2(\mathbb{R})$ and $L^2(\mathbb{R}) \not\subset L^1(\mathbb{R})$).
- **Theorem.** If $f \in L^2(\mathbb{R})$ and f is compactly supported, then $f \in L^1(\mathbb{R})$.
- **Theorem.** $L^1(\mathbb{R}) \cap L^2(\mathbb{R})$ is dense in $L^2(\mathbb{R})$.

Equality in L^p Spaces

- Consider $x_1, x_2 \in L^p(\mathbb{R})$, where $x_1(t) = 0$ and $x_2(t) = \begin{cases} 1 & \text{if } t = 0 \\ 0 & \text{otherwise.} \end{cases}$
- Clearly, x_1 and x_2 are ***not pointwise equal*** (i.e., they are not equal at every point).
- Since $x_1(t) = x_2(t)$ except for $t = 0$, $x_1 - x_2 = 0$ almost everywhere. Thus, $|x_1 - x_2| = 0$ almost everywhere.
- Consider the norm of $x_1 - x_2$, which is given by $\|x_1 - x_2\|_{L^p} = \left[\int_{\mathbb{R}} |x_1(t) - x_2(t)|^p \, dt\right]^{1/p}$.
- Since $|x_1 - x_2|$ is zero almost everywhere, the preceding integral is zero. Thus, $\|x_1 - x_2\| = 0$.
- From the properties of a norm, $\|x_1 - x_2\| = 0$ implies that $x_1 - x_2 = 0$, meaning that $x_1 = x_2$.
- To avoid contradictions, x_1 and x_2 are defined to be equal if they ***differ only on a set of measure zero*** (i.e., $x_1 = x_2$ almost everywhere), since $x_1 = x_2 \Leftrightarrow x_1 - x_2 = 0 \Leftrightarrow \|x_1 - x_2\| = 0 \Leftrightarrow |x_1 - x_2| =$ 0 almost everywhere $\Leftrightarrow x_1 = x_2$ almost everywhere.

l^p Spaces

- The $l^p(I)$ spaces are a family of complete normed (i.e., Banach) spaces, where the parameter $p \in [1,\infty) \cup \{\infty\}$.
- The underlying set is comprised of all complex (or real) ***sequences*** $(x_n)_{n \in I}$ such that

$$\begin{cases} \sum_{n \in I} |x_n|^p < \infty & p \in [1,\infty) \\ \sup_{n \in I} |x_n| < \infty & p = \infty. \end{cases}$$

- The cases of $p = 1$, $p = 2$, and $p = \infty$ correspond to the spaces of absolutely-summable, square-summable, and bounded sequences, respectively. The set I is often $\mathbb{Z}$ or a subset of $\mathbb{Z}$ such as $\mathbb{Z}^*$.
- Vector addition and scalar multiplication are defined in the straightforward way for sequences.
- The norm employed is given by

$$\|x\|_p = \begin{cases} \left[\sum_{n \in I} |x_n|^p\right]^{1/p} & p \in [1,\infty) \\ \sup_{n \in I} |x_n| & p = \infty. \end{cases}$$

l^p Spaces (Continued)

- When $p = 2$, we obtain a Hilbert space (i.e., $l^2(I)$), where the inner product is given by

$$\langle x, y \rangle = \sum_{n \in I} x_n y_n^*.$$

- The $l^p(I)$ spaces are nested as follows: $l^1(I) \subset l^2(I) \subset l^\infty(I)$.

Matrices

- The transpose, conjugate transpose, and transposed inverse of a matrix $\boldsymbol{A}$ are denoted, respectively, as $\boldsymbol{A}^T$, $\boldsymbol{A}^\dagger$, and $\boldsymbol{A}^{-T}$.
- The symbols $\boldsymbol{I}_n$ and $\boldsymbol{J}_n$ denote the $n \times n$ identity and anti-identity matrices, respectively, where the subscript n may be omitted when clear from the context (e.g., $\boldsymbol{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ and $\boldsymbol{J}_3 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$).
- The (k,l)th **minor** of the $n \times n$ matrix $\boldsymbol{A}$ is the determinant of the $(n-1) \times (n-1)$ matrix formed by removing the kth row and lth column from $\boldsymbol{A}$.
  Example. The $(2,2)$th minor of $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ is $\det \begin{bmatrix} 1 & 3 \\ 7 & 9 \end{bmatrix}$.
- The **adjugate** of the $n \times n$ matrix $\boldsymbol{A}$, denoted $\operatorname{Adj}\boldsymbol{A}$, is the $n \times n$ matrix whose (k,l)th entry is given by $(-1)^{k+l} M_{l,k}$ where $M_{l,k}$ is the (l,k)th minor of $\boldsymbol{A}$.
  Example. $\operatorname{Adj} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} (1)(4) & (-1)(3) \\ (-1)(2) & (1)(1) \end{bmatrix}^T = \begin{bmatrix} 4 & -2 \\ -3 & 1 \end{bmatrix}$.

Matrices, Divisibility, Roots of Unity

- **Cramer's rule.** The inverse of a square matrix $\boldsymbol{A}$ is given by $\boldsymbol{A}^{-1} = \frac{1}{\det \boldsymbol{A}} \operatorname{Adj} \boldsymbol{A}$.
  Example. Let $\boldsymbol{A} = \left[\begin{smallmatrix} 1 & 2 \\ 3 & 4 \end{smallmatrix}\right]$. Then,
  $\boldsymbol{A}^{-1} = \frac{1}{\det \boldsymbol{A}} \operatorname{Adj} \boldsymbol{A} = -\frac{1}{2} \left[\begin{smallmatrix} 4 & -2 \\ -3 & 1 \end{smallmatrix}\right] = \begin{bmatrix} -2 & 1 \\ \frac{3}{2} & -\frac{1}{2} \end{bmatrix}$.
- A square matrix $\boldsymbol{A}$ is said to be **unitary** if $\boldsymbol{A}^{-1} = \boldsymbol{A}^{\dagger}$.
- For a polynomial matrix $\boldsymbol{A}(z)$, the notation $\boldsymbol{A}_*(z)$ denotes the matrix obtained by conjugating the polynomial coefficients without conjugating z.
- If a square matrix $\boldsymbol{A}(z)$ is such that $\boldsymbol{A}(z)\boldsymbol{A}_*^T(z^{-1}) = \boldsymbol{I}$, then $\boldsymbol{A}$ is said to be **paraunitary**.
- For $a, b \in \mathbb{Z}$, we say that ***a divides b***, abbreviated $a \mid b$, if there exists $k \in \mathbb{Z}$ such that $b = ka$ (i.e., b is an integer multiple of a).
  Example. $2 \mid 4$ and $3 \mid 27$, while $7 \nmid 13$.
- The Mth root of unity given by $e^{-j2\pi/M}$ is denoted as W_M.

Toeplitz Matrices

- A $n \times n$ matrix with each diagonal having entries of equal value is said to be **Toeplitz**. In other words, such a matrix has the form

$$\begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{n-1} \\ a_{-1} & a_0 & a_1 & \cdots & a_{n-2} \\ a_{-2} & a_{-1} & a_0 & \cdots & a_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{-(n-1)} & a_{-(n-2)} & a_{-(n-3)} & \cdots & a_0 \end{bmatrix}.$$

- Toeplitz matrices have a special significance in the context of convolution.

Section 2.8

Fourier Analysis

Fourier Transform in $L^1(\mathbb{R})$

- The classical definition of the Fourier transform, as taught in most introductory signal-processing courses, is only applicable to functions in $L^1(\mathbb{R})$.
- For $f \in L^1(\mathbb{R})$, the **Fourier transform** of f, denoted $\hat{f}$, is $\hat{f}(\omega) = \int_{\mathbb{R}} f(t)e^{-j\omega t}dt$.
- If $f, \hat{f} \in L^1(\mathbb{R})$, then the **inverse Fourier transform** f of $\hat{f}$ is given by $f(t) = \frac{1}{2\pi}\int_{\mathbb{R}} \hat{f}(\omega)e^{j\omega t}d\omega$.
- It can be shown that integral in the above definition of the Fourier transform converges if and only if $f \in L^1(\mathbb{R})$. This is why we have the restriction that $f \in L^1(\mathbb{R})$. A similar issue also arises with the integral for the inverse Fourier transform.
- Since some functions in $L^2(\mathbb{R})$ are not in $L^1(\mathbb{R})$ (e.g., the sinc function), the Fourier transform as described above is not well defined for all functions in $L^2(\mathbb{R})$.
- Unfortunately, in engineering, we are usually more interested in using $L^2(\mathbb{R})$ than $L^1(\mathbb{R})$.

Defining the Fourier Transform in $L^2(\mathbb{R})$

- The classical definition of the Fourier transform has to be modified in order to be well defined for all functions in $L^2(\mathbb{R})$.
- Since $L^1(\mathbb{R}) \cap L^2(\mathbb{R})$ is dense in $L^2(\mathbb{R})$, any $f \in L^2(\mathbb{R})$ can be expressed as $f = \lim_{n\to\infty} f_n$, where $f_n \in L^1(\mathbb{R}) \cap L^2(\mathbb{R})$ and $n \in \mathbb{N}$. For example, we can choose $f_n(t) = f(t)\,\mathrm{rect}(\frac{t}{2n})$.
- Since each $f_n \in L^1(\mathbb{R})$, the $L^1(\mathbb{R})$ Fourier transform of f_n exists.
- Therefore, a natural way to define the Fourier transform for $L^2(\mathbb{R})$ is as the limit of $(\hat{f}_n)_{n\in\mathbb{N}}$:

$$\begin{aligned}\hat{f}(\omega) = \lim_{n\to\infty} \hat{f}_n(\omega) &= \lim_{n\to\infty} \int_{\mathbb{R}} f_n(t) e^{-j\omega t} dt \\ &= \lim_{n\to\infty} \int_{\mathbb{R}} f(t)\,\mathrm{rect}(\tfrac{t}{2n}) e^{-j\omega t} dt \\ &= \lim_{n\to\infty} \int_{-n}^{n} f(t) e^{-j\omega t} dt\end{aligned}$$

Fourier Transform in $L^2(\mathbb{R})$

- **Fourier transform.** For $f \in L^2(\mathbb{R})$, $\hat{f}(\omega) = \lim_{n\to\infty} \int_{-n}^{n} f(t)e^{-j\omega t}dt$.
- **Inverse Fourier transform.** For $f \in L^2(\mathbb{R})$, $f(t) = \lim_{n\to\infty} \frac{1}{2\pi} \int_{-n}^{n} \hat{f}(\omega)e^{j\omega t}d\omega$.
- The $L^2(\mathbb{R})$ definition of the Fourier transform shares many of the same basic properties as the $L^1(\mathbb{R})$ definition, including:
  - $[af+bg]\hat{} = a\hat{f} + b\hat{g}$ (**linearity**)
  - $[f(\cdot - t_0)]\hat{}(\omega) = e^{-j\omega t_0}\hat{f}(\omega)$ (**translation**)
  - $[e^{j\omega_0\cdot}f(\cdot)]\hat{}(\omega) = \hat{f}(\omega - \omega_0)$ (**modulation**)
  - $[f(a\cdot)]\hat{}(\omega) = \frac{1}{|a|}\hat{f}(\omega/a)$ (**scaling**)
  - $[f^*(\cdot)]\hat{}(\omega) = \hat{f}^*(-\omega)$ (**conjugation**)
  - $[Df]\hat{} = j\omega\hat{f}$, where D denotes the derivative operator (**differentiation**)
  - $[f*g]\hat{} = \hat{f}\hat{g}$ (**convolution**)
- **Moment property.** The kth moment μ_k of a function f is given by $\mu_k = j^k\hat{f}^{(k)}(0)$, where $\hat{f}^{(k)}$ denotes the kth order derivative of $\hat{f}$.

Fourier Transform in $L^2(\mathbb{R})$ (Continued)

- **Parseval relation.** If $f, g \in L^2(\mathbb{R})$, then $\langle f, g \rangle = \frac{1}{2\pi} \langle \hat{f}, \hat{g} \rangle$ (i.e., the Fourier transform preserves inner products up to a scale factor).
- **Plancherel.** If $f \in L^2(\mathbb{R})$, then $\hat{f} \in L^2(\mathbb{R})$ and $\|f\|^2 = \frac{1}{2\pi} \left\|\hat{f}\right\|^2$ (i.e., the Fourier transform preserves norms up to a scale factor).
- Since the Fourier transform preserves norms (up to scale), $L^2(\mathbb{R})$ is closed under Fourier transform and inverse Fourier transform operations.
- **Riemann-Lebesgue lemma.** If $f \in L^1(\mathbb{R})$, then $\hat{f}$ is continuous on $\mathbb{R}$ and $\lim_{|\omega| \to \infty} \hat{f}(\omega) = 0$.

Part 3

One-Dimensional Multirate Filter Banks

Multirate Signal Processing

- Signal processing that deals with signals sampled at a single rate is said to be **unirate**, while signal processing that deals with signals sampled at more than one sampling rate is said to be **multirate**.
- Systems can be classified as unirate or multirate depending on whether they deal with signals sampled at only one or more than one rate.
- Multirate systems are extremely useful in a wide variety of applications.
- Sometimes, a multirate system can be used to perform a task ***more easily*** or ***more efficiently*** than is possible with a unirate system (e.g., by performing some computations at a lower sampling rate).
- Other times, a task may ***inherently require*** the use of multiple sampling rates (e.g., sampling rate conversion).
- In addition to all of the familiar operations associated with unirate signal processing (such as convolution and delay/advance), multirate signal processing also defines operations for ***changing the sampling rate***.
- Changing the sampling rate is accomplished via operations known as ***downsampling*** and ***upsampling***.

Section 3.1

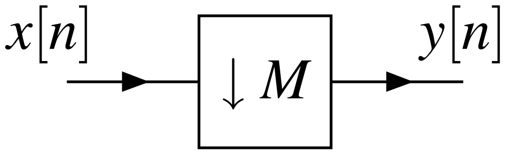
Multirate Fundamentals

Downsampling

- In multirate signal processing, the basic operation for ***decreasing*** the sampling rate is known as **downsampling** and is performed by a **downsampler**.
- The **M-fold downsampling** operation takes an input sequence x and produces the output sequence y as given by

  $$y[n] = (\downarrow M)x[n] = x[Mn],$$

  where M is an integer.
- The constant M is referred to as the **downsampling factor**.
- The M-fold downsampler, which embodies the M-fold downsampling operation, is depicted as shown below.



- In simple terms, the downsampling operation ***keeps every Mth sample*** and discards the others.
- Downsampling is a ***linear (periodically) time-varying*** operation.

Downsampling Example

- Consider the sequence $x = (x_n)_{n \in \mathbb{Z}}$. That is, x is as shown below.

n	$\cdots$	-4	-3	-2	-1	0	1	2	3	4	$\cdots$
$x[n]$	$\cdots$	x_{-4}	x_{-3}	x_{-2}	x_{-1}	x_0	x_1	x_2	x_3	x_4	$\cdots$

- The sequence $(\downarrow 2)x$ is as shown below.

n	$\cdots$	-3	-2	-1	0	1	2	3	$\cdots$
$(\downarrow 2)x[n]$	$\cdots$	x_{-6}	x_{-4}	x_{-2}	x_0	x_2	x_4	x_6	$\cdots$

- The sequence $(\downarrow 3)x$ is as shown below.

n	$\cdots$	-3	-2	-1	0	1	2	3	$\cdots$
$(\downarrow 3)x[n]$	$\cdots$	x_{-9}	x_{-6}	x_{-3}	x_0	x_3	x_6	x_9	$\cdots$

Downsampling in $\mathcal{Z}$ Domain

- **Theorem.** Let x and y be sequences related by $y = (\downarrow M)x$. Let X and Y denote the $\mathcal{Z}$ transforms of x and y, respectively. Then, Y, which is denoted as $(\downarrow M)X$, is given by

  $$Y(z) = (\downarrow M)X(z) = \frac{1}{M}\sum_{k=0}^{M-1} X(z^{1/M}W_M^k),$$

  where $W_M = e^{-j2\pi/M}$.
- The above summation is carefully constructed so that all terms with non-integer powers of z cancel (i.e., add to zero).
- So, although at first glance, the above expression may appear to contain non-integer powers of z, this is not the case.

Downsampling in Fourier Domain

- **Theorem.** Let x and y be sequences related by $y = (\downarrow M)x$. Let $\hat{x}$ and $\hat{y}$ denote the Fourier transforms of x and y, respectively. If the sampling period before and after downsampling is normalized to one, the following relationship holds:

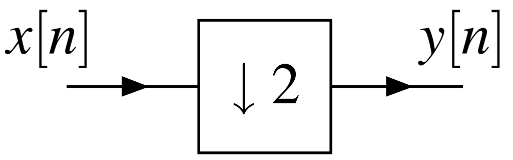
$$\hat{y}(\omega) = \frac{1}{M} \sum_{k=0}^{M-1} \hat{x}\left(\frac{\omega - 2\pi k}{M}\right).$$

- Note that $\hat{x}\left(\frac{\cdot - 2\pi k}{M}\right)$ is $\hat{x}$ first dilated by $1/M$ and then translated by $2\pi k$.
- The spectrum of the downsampled signal is merely the ***average*** of M translated copies of the original input spectrum.
- Due to our convention of normalizing the sampling period after downsampling to one, the spectrum is also ***dilated***.
- It is important to understand, however, that this spectrum dilation effect is only a consequence of the sampling-period renormalization and is ***not caused by downsampling itself***.

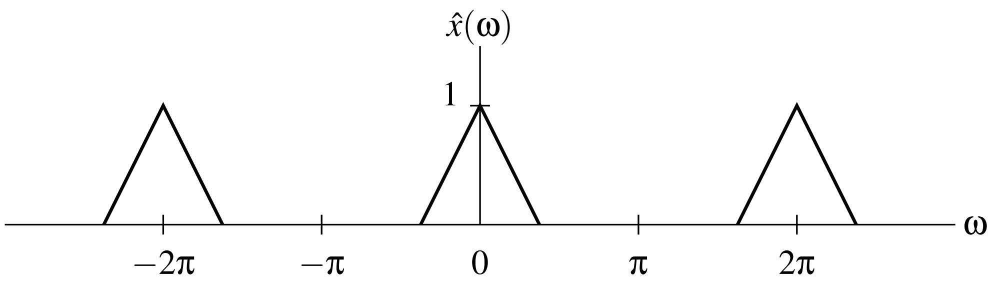
Downsampling and Aliasing

- Downsampling can result in multiple baseband frequencies in the input signal being mapped to a single frequency in the output signal.
- This phenomenon is called **aliasing**.
- When aliasing occurs, information in the signal being downsampled is ***irretrievably lost*** and the effects of downsampling cannot be undone.
- When a signal x is downsampled by a factor of M, aliasing ***cannot occur*** if x is bandlimited to frequencies ω satisfying $|\omega| < \pi/M$.

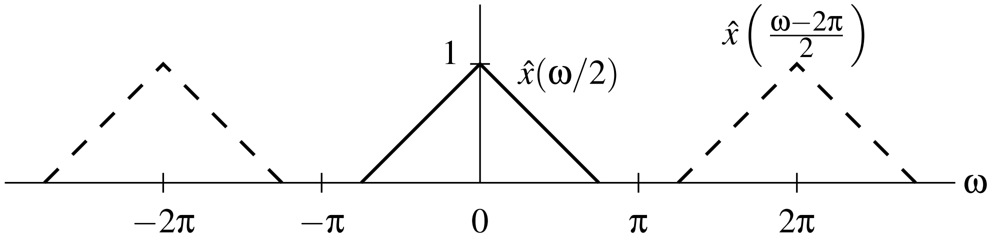
Downsampling Example: No Aliasing Case



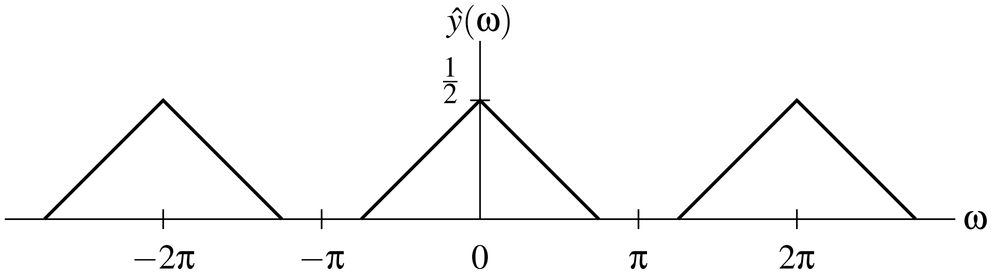
2-Fold Downsampler



Spectrum of Input Signal

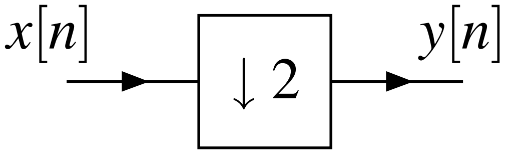


Translated Copies of Input Spectrum

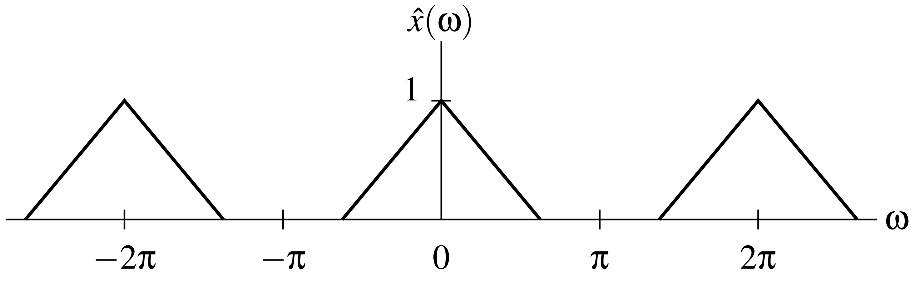


Spectrum of Downsampled Signal

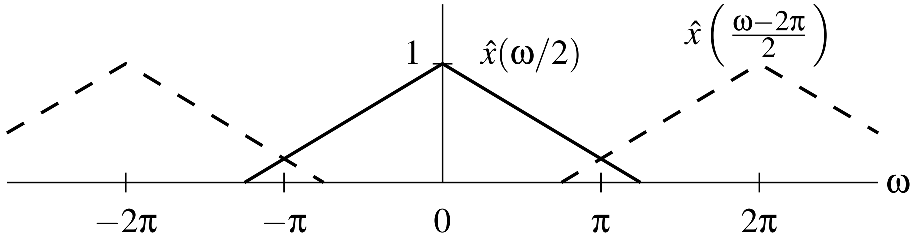
Downsampling Example: Aliasing Case



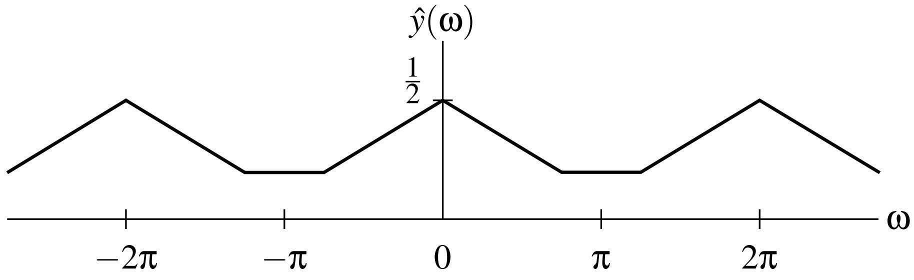
2-Fold Downsampler



Spectrum of Input Signal



Translated Copies of Input Spectrum



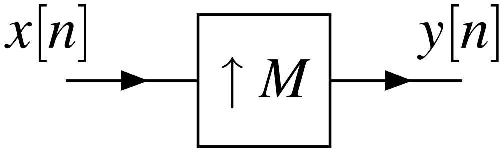
Spectrum of Downsampled Signal

Upsampling

- In multirate signal processing, the basic operation for ***increasing*** the sampling rate is known as **upsampling** and is performed by an **upsampler**.
- The M**-fold upsampling** operation takes an input sequence x and produces the output sequence y as given by

$$y[n] = (\uparrow M)x[n] = \begin{cases} x[n/M] & \text{if } M \mid n \\ 0 & \text{otherwise,} \end{cases}$$

  where M is an integer (and "$M \mid n$" means $\frac{n}{M} \in \mathbb{Z}$).
- The M-fold upsampler, which embodies the M-fold upsampling operation, is depicted as shown below.



- The constant M is referred to as the **upsampling factor**.
- In simple terms, the upsampling operation inserts $M-1$ zeros between the samples of the original signal.
- Upsampling is a ***linear (periodically) time-varying*** operation.

Upsampling Example

- Consider the sequence $x = (x_n)_{n \in \mathbb{Z}}$. That is, x is as shown below.

n	$\cdots$	-4	-3	-2	-1	0	1	2	3	4	$\cdots$
$x[n]$	$\cdots$	x_{-4}	x_{-3}	x_{-2}	x_{-1}	x_0	x_1	x_2	x_3	x_4	$\cdots$

- The sequence $(\uparrow 2)x$ is as shown below.

n	$\cdots$	-4	-3	-2	-1	0	1	2	3	4	$\cdots$
$(\uparrow 2)x[n]$	$\cdots$	x_{-2}	0	x_{-1}	0	x_0	0	x_1	0	x_2	$\cdots$

- The sequence $(\uparrow 3)x$ is as shown below.

n	$\cdots$	-3	-2	-1	0	1	2	3	$\cdots$
$(\uparrow 3)x[n]$	$\cdots$	x_{-1}	0	0	x_0	0	0	x_1	$\cdots$

Upsampling in $\mathcal{Z}$ Domain

- **Theorem.** Let x and y be sequences related by $y = (\uparrow M)x$. Let X and Y denote the $\mathcal{Z}$ transforms of x and y, respectively. Then, Y, which we denote as $(\uparrow M)X$, is given by

$$Y(z) = (\uparrow M)X(z) = X(z^M).$$

Upsampling in Fourier Domain

- **Theorem.** Let x and y be sequences related by $y = (\uparrow M)x$. Let $\hat{x}$ and $\hat{y}$ denote the Fourier transforms of x and y, respectively. Assuming that the sampling period before and after upsampling is normalized to one, this directly yields the relationship

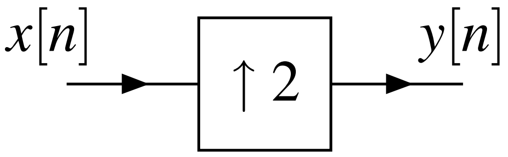
$$\hat{y}(\omega) = \hat{x}(M\omega).$$

- The upsampling process simply serves to ***move the location of the sampling frequency*** on the frequency axis.
- Due to our convention of normalizing the sampling period after upsampling to one, the spectrum is also ***dilated***.
- It is important to understand, however, that this compression effect is only a consequence of the sampling period renormalization and is ***not caused by upsampling itself***.

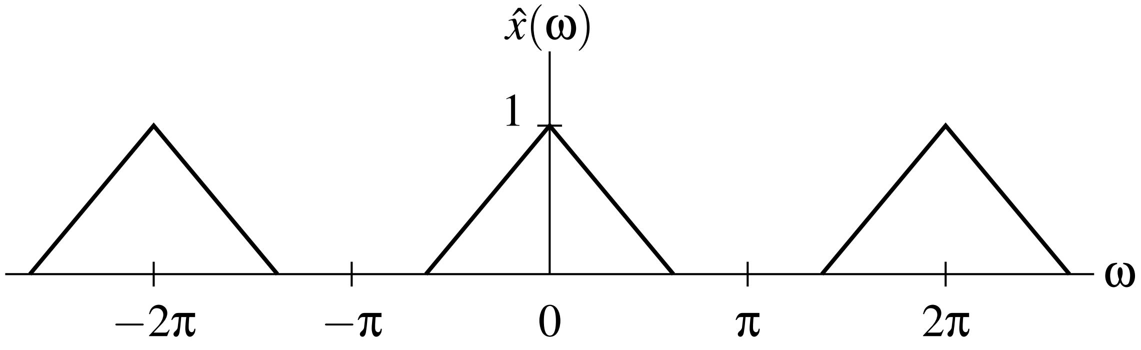
Upsampling and Imaging

- Since the shape of the spectrum is not altered by upsampling, there is ***no information loss*** and the original signal can always be recovered from its upsampled version.
- Upsampling, however, does result in the creation of ***multiple copies*** of the original baseband spectrum.
- This phenomenon is called **imaging**.
- These copies of the baseband spectrum are referred to as **images**.

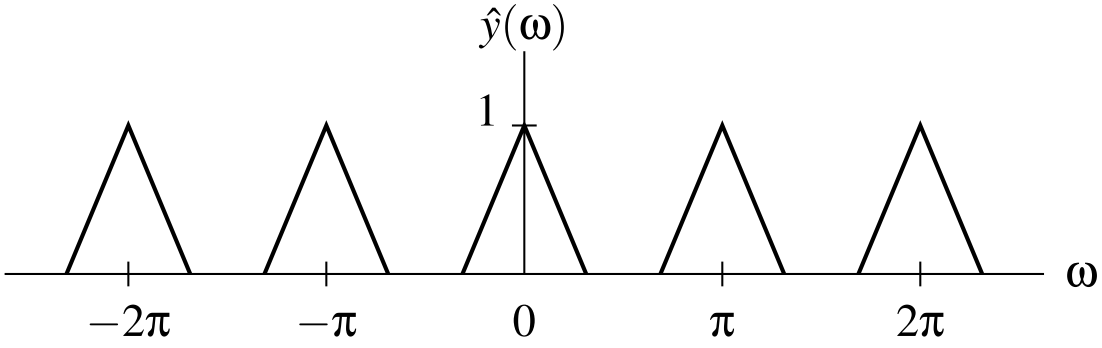
Upsampling Example



2-Fold Upsampler



Spectrum of Input Signal



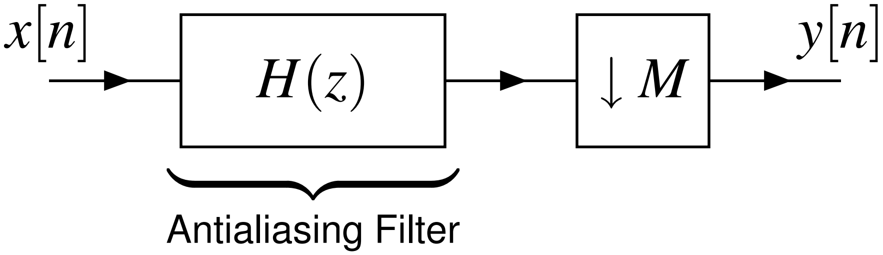
Spectrum of Upsampled Signal

Decimation

- Consider ***decreasing the sampling rate*** by an integer factor M.
- Although this could be accomplished using only an M-fold downsampler, such a scheme has a major shortcoming, namely that ***severe aliasing*** can result (which is highly undesirable in most applications).
- To avoid (or minimize) aliasing, we apply a ***lowpass filter*** to the signal prior to downsampling.
- This filter is chosen to ensure that the signal being downsampled is sufficiently bandlimited so as to ***avoid (or minimize) aliasing***.
- The lowpass filter using in this context is called an **antialiasing filter**.
- The ***cutoff frequency*** of the antialiasing filter is π/M.
- The ***passband gain*** of the antialiasing filter is one.
- The above approach leads to what is known as **M-fold decimation** and is performed by an **M-fold decimator** (i.e., an antialiasing filter followed by an M-fold downsampler).

Decimation (Continued)

- An M-fold decimator has the form shown below.



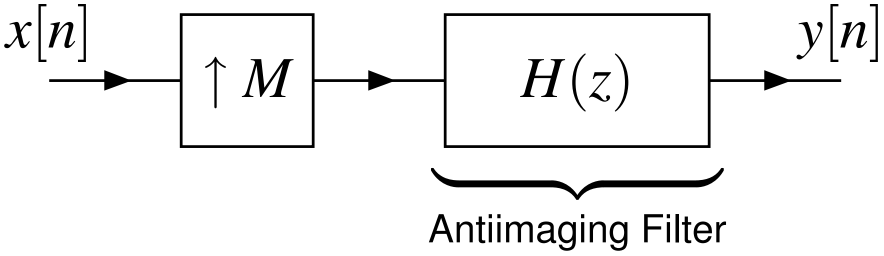
- In practice, an M-fold decimator is not implemented directly using the computational structure shown above, since it is ***very computationally inefficient***.
- That is, due to downsampling, most of elements of the sequence output from the antialiasing filter are discarded.
- We will later explore more efficient computational structures for decimation.

Interpolation

- Consider ***increasing the sampling rate*** by an integer factor M.
- Although this could be done using only an M-fold upsampler, such a scheme has a ***major shortcoming***, namely that upsampling results in ***imaging*** (which is highly undesirable in most applications).
- Consequently, we usually introduce a ***lowpass filtering*** operation after upsampling to eliminate images of the original baseband spectrum.
- The lowpass filter used in this context is called an **antiimaging filter**.
- The ***cutoff frequency*** of the antiimaging filter is π/M.
- The ***passband gain*** of the antiimaging filter is M.
- The above approach leads to what is known as **M-fold interpolation** and is performed by an **M-fold interpolator** (i.e., an M-fold upsampler followed by an antiimaging filter).

Interpolation (Continued)

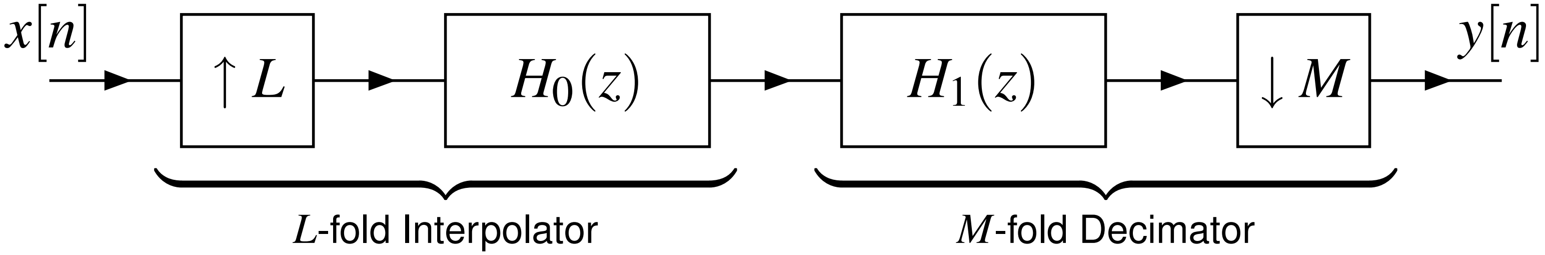
- An M-fold interpolator has the form shown below.



- In practice, an M-fold interpolator is not implemented directly using the computational structure shown above, since it is ***very computationally inefficient***.
- That is, due to upsampling, the input sequence to the antiimaging filter is mostly zero, resulting in many additions/multiplications involving zero during convolution.
- We will later explore more computationally efficient means for implementing interpolation.

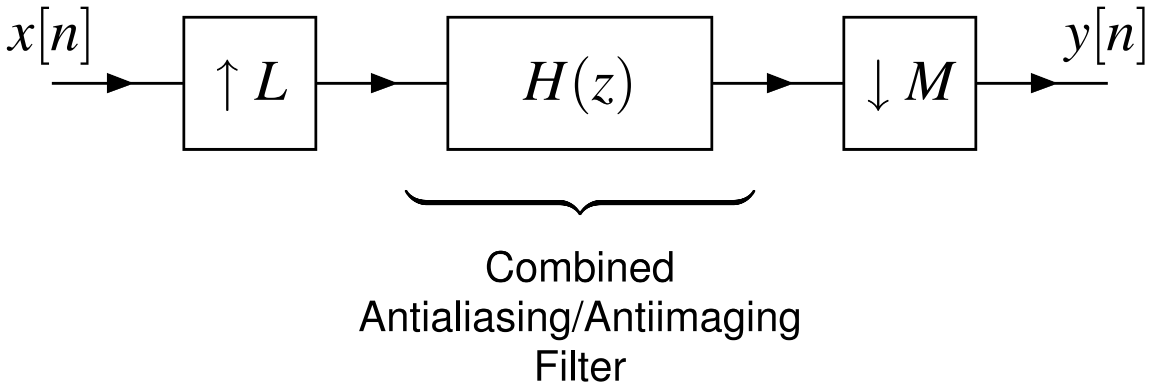
Rational Sampling-Rate Conversion

- In some situations, we may need to change the sampling rate by a ***noninteger*** factor.
- In particular, we sometimes want to change the sampling rate by a ***rational factor*** L/M.
- This can be accomplished by applying L-fold interpolation followed by M-fold decimation.
- This leads to the rational sampling-rate converter shown below.



Rational Sampling-Rate Conversion (Continued)

- The antiimaging and antialiasing filtering operations associated with interpolation and decimation can be ***combined*** into a single filtering operation, leading to the rational sampling-rate converter shown below.



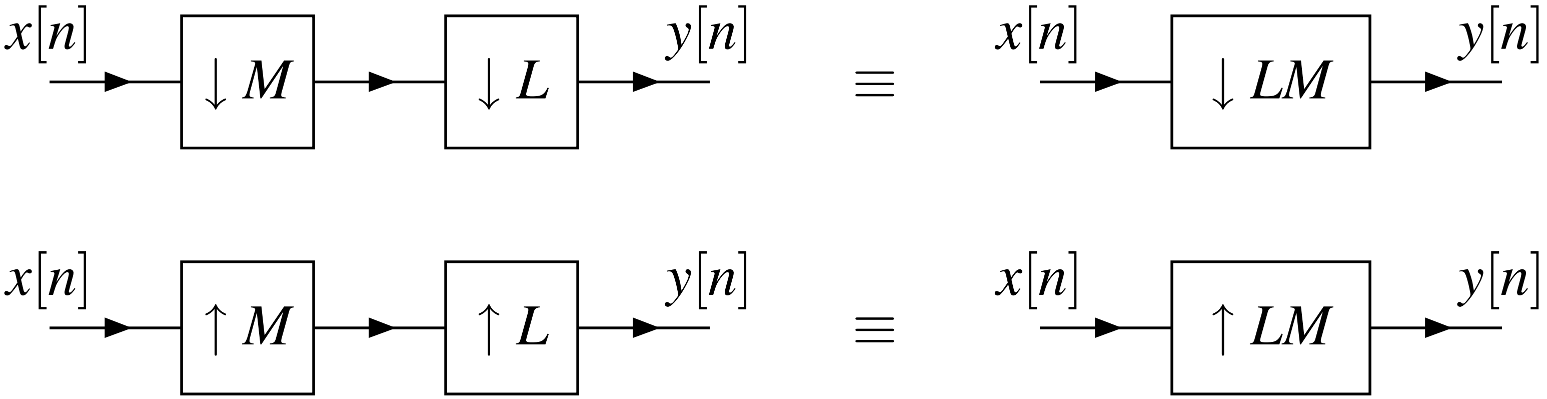
- In practice, a rational sampling-rate converter is not implemented directly using the computational structure shown above, since it is ***very computationally inefficient***.
- We will later explore more computationally efficient means for implementing rational sampling rate converters.

Cascaded Upsampling and Downsampling Identities

- Often, multiple upsampling operations or multiple downsampling operations may be applied in succession. Thus, we would like to consider the effect of cascading operations in this manner.
- **Theorem.** The downsampling and upsampling operators have the following properties:

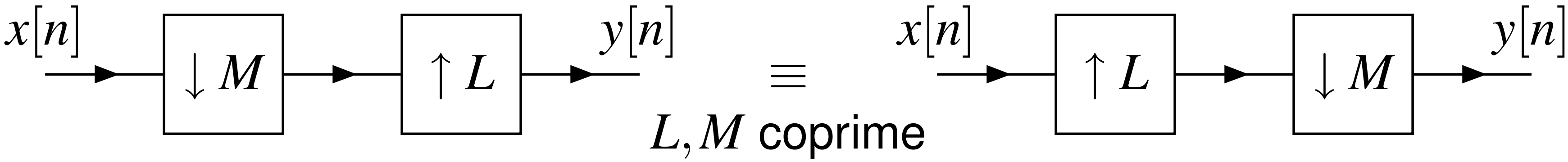
$$(\downarrow M)(\downarrow L) = \downarrow LM = \downarrow ML \quad \text{and}$$
$$(\uparrow M)(\uparrow L) = \uparrow LM = \uparrow ML.$$

- In other words, we have the identities shown below.



Commutativity of Upsampling and Downsampling

- One might wonder if upsampling and downsampling commute.
- These operations only commute under certain circumstances as given by the result below.
- **Theorem.** The L-fold upsampling and M-fold downsampling operators commute (i.e., $(\uparrow L)(\downarrow M) = (\downarrow M)(\uparrow L)$) if and only if L and M are coprime.
- In other words, we have the identity shown below.



- The above relationship has both theoretical and practical utility. It can sometimes be used to ***simplify expressions*** involving upsampling and downsampling operations, and also may be used to ***obtain more desirable implementations*** of multirate systems in some situations.

Noble Identities

- Often a downsampler or upsampler appears in cascade with a filter.
- Although it is not always possible to ***interchange the order of upsampling/downsampling and filtering*** without changing system behavior, it is sometimes possible to find an equivalent system with the order of these operations reversed, through the use of two very important relationships called the noble identities.
- In addition to their ***theoretical utility***, the noble identities are of ***great practical significance***.
- For performance reasons, it is usually desirable to perform filtering operations on the side of an upsampler (or downsampler) with the lower sampling rate.
- Using the noble identities, we can move filtering operations across upsamplers (or downsamplers) and achieve improved computational efficiency.

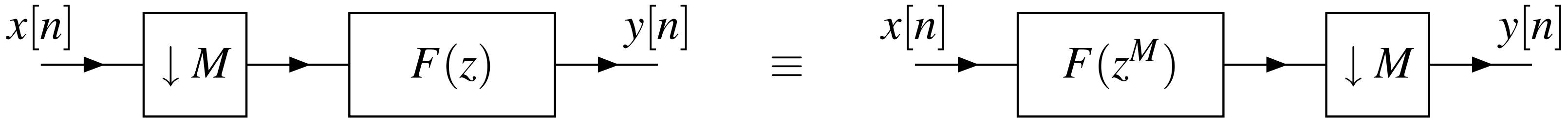
First Noble Identity

- **First noble identity.** For any two sequences with $\mathcal{Z}$ transforms X and F, the following identity holds:

  $$F(z)\left[(\downarrow M)X(z)\right] = (\downarrow M)\left[F(z^M)X(z)\right],$$

  where $F(z)$ is a ***rational*** polynomial.

- In other words, we have the identity shown below.



$x[n]$ → $\downarrow M$ → $F(z)$ → $y[n]$ $\equiv$ $x[n]$ → $F(z^M)$ → $\downarrow M$ → $y[n]$

- The first noble identity allows us to replace a filtering operation on one side of a downsampler with an equivalent filtering operation on the other side of the downsampler.

- It is important to emphasize that, in order for the above identity to hold, $F(z)$ must be a ***rational*** polynomial.

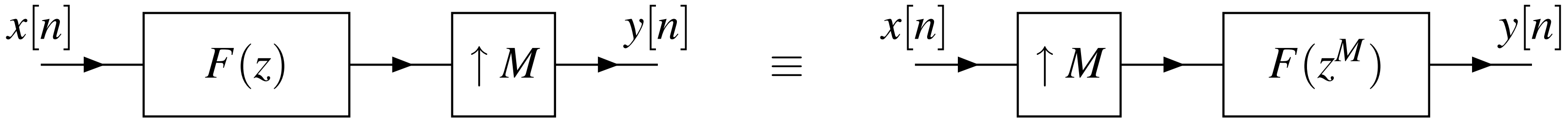
Second Noble Identity

- **Second noble identity.** For any two sequences with $\mathcal{Z}$ transforms X and F, the following identity holds:

$$(\uparrow M)\left[F(z)X(z)\right] = F(z^M)\left[(\uparrow M)X(z)\right].$$

  where $F(z)$ is a ***rational*** polynomial.

- In other words, we have the identity shown below.



- The second noble identity allows us to replace a filtering operation on one side of an upsampler with an equivalent filtering operation on the other side of the upsampler.
- It is important to emphasize that, in order for the above identity to hold, $F(z)$ must be a ***rational*** polynomial.

Polyphase Representations

- The **polyphase representation** of the signal $x[n]$, with respect to an integer M and a set of integers $\{m_k\}_{k=0}^{M-1}$, is defined as
$$x[n] = \sum_{k=0}^{M-1} ((\uparrow M)x_k)[n+m_k],$$
where
$$x_k[n] = (\downarrow M)(x[n-m_k]) = x[Mn-m_k]$$
and the set $\{m_k\}_{k=0}^{M-1}$ is chosen such that
$$\operatorname{mod}(m_k, M) \neq \operatorname{mod}(m_l, M) \text{ whenever } k \neq l.$$
- As a matter of terminology, we refer to M as a **sampling factor** (or the number of phases), the elements of the set $\{m_k\}_{k=0}^{M-1}$ as **coset offsets**, and $x_0[n], x_1[n], \ldots, x_{M-1}[n]$ as **polyphase components**.
- A polyphase representation partitions the samples of the original sequence x into M subsequences $\{x_k\}_{k=0}^{M-1}$.
- Polyphase representations are of fundamental importance in multirate signal processing.

Choice of Polyphase Representation

- A sequence has ***infinitely many*** polyphase representations, since infinitely many choices exist for the sampling factor M and coset offsets $\{m_k\}_{k=0}^{M-1}$.
- Even for a fixed choice of M, the polyphase representation is not uniquely determined, since more than one choice for the $\{m_k\}_{k=0}^{M-1}$ is possible.
- Although, for a given sampling factor M, many different choices are possible for the $\{m_k\}_{k=0}^{M-1}$, four specific choices are ***most frequently used*** in practice, referred to as type 1, 2, 3, and 4.
- In the case of these four commonly used types of polyphase representations, the $\{m_k\}_{k=0}^{M-1}$ are chosen as

$$m_k = \begin{cases} -k & \text{type 1} \\ k-(M-1) & \text{type 2} \\ k & \text{type 3} \\ (M-1)-k & \text{type 4.} \end{cases}$$

Choice of Polyphase Representation (Continued)

- For a given choice of M, one can show that different choices of $\{m_k\}_{k=0}^{M-1}$ serve only to ***time shift and permute*** the polyphase components.
- For example, for a fixed choice of M, the type-2 polyphase components of a sequence are simply a permutation of its type-1 polyphase components. More specifically, the orders of the components are reversed with respect to one another.
- The particular type of polyphase representation to be used is normally dictated by practical considerations or notational convenience.

Polyphase Representation in $\mathcal{Z}$ Domain

- Since we often work with $\mathcal{Z}$ transforms of sequences, it is convenient to express the polyphase representation in the $\mathcal{Z}$ domain.
- **Theorem.** Let X denote the $\mathcal{Z}$ transform of x, and let X_k denote the $\mathcal{Z}$ transform of x_k for $k = 0, 1, \ldots, M-1$. Expressed in the $\mathcal{Z}$ domain, the polyphase representation of x is given by

$$X(z) = \sum_{k=0}^{M-1} z^{m_k} X_k(z^M),$$

where

$$X_k(z) = (\downarrow M) z^{-m_k} X(z).$$

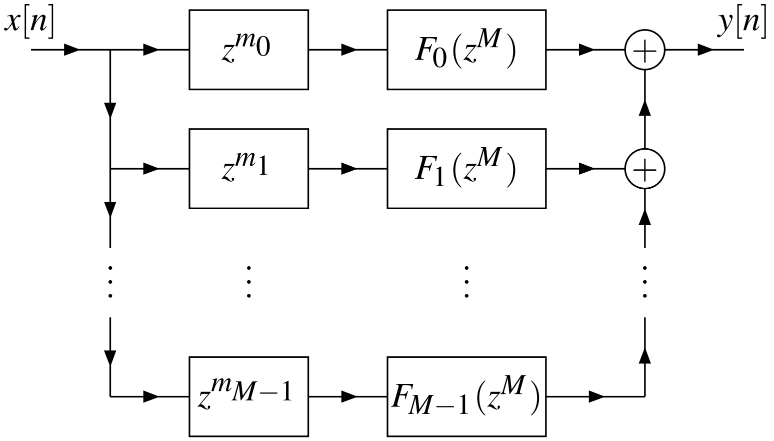
- In passing, we note that if X is rational (but not a Laurent polynomial), the polyphase components need not be rational.

Polyphase Representations of Filters

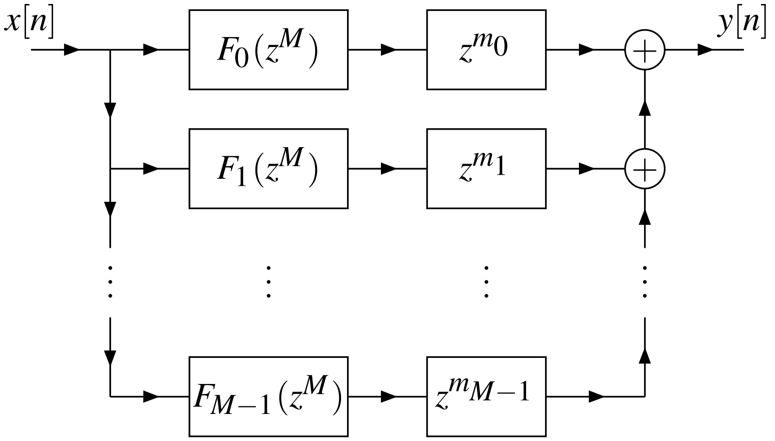
- We can express the transfer function F of a filter in terms of a polyphase representation as

$$F(z) = \sum_{k=0}^{M-1} z^{m_k} F_k(z^M).$$

- This representation suggests an implementation strategy for a filter known as the polyphase realization, as shown below.



First Variant



Second Variant

- The original filter is implemented using M filters, each having a transfer function that is a rational polynomial in z^M.

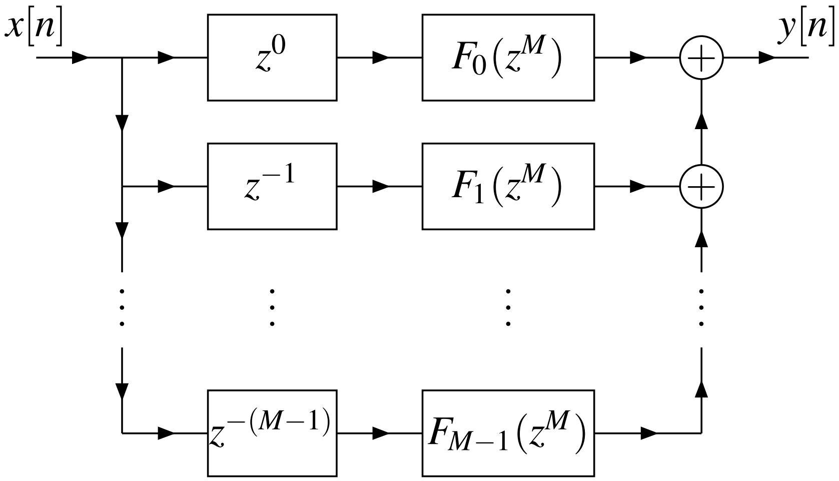
Polyphase Representations of Filters (Continued)

- The polyphase representation is often a mathematically convenient form in which to express filtering operations in multirate systems, simplifying many theoretical results.
- Perhaps, more importantly, the polyphase representation leads to an efficient means for implementing filtering operations in a multirate framework.
- For type 1, 2, 3, and 4 polyphase representations, the coset offsets $\{m_k\}_{k=0}^{M-1}$ are chosen to be consecutive integers.

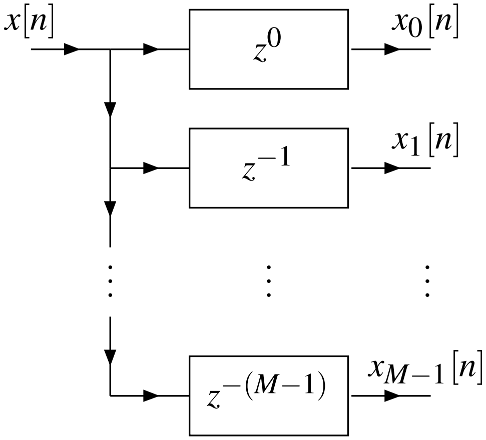
Type	m_k Values
1	$0, -1, \ldots, -(M-1)$
2	$-(M-1), -(M-2), \ldots, 0$
3	$0, 1, \ldots, M-1$
4	$M-1, M-2, \ldots, 0$

- This allows all of the delays/advances to be implemented with a chain of M unit-delays/unit-advances.

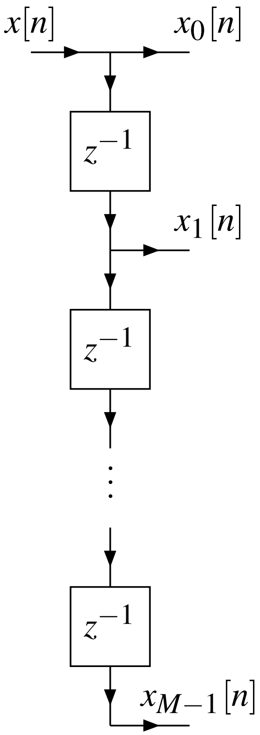
Type-1 Polyphase Representation and Advance/Delay Chains



Type-1 Polyphase Representation

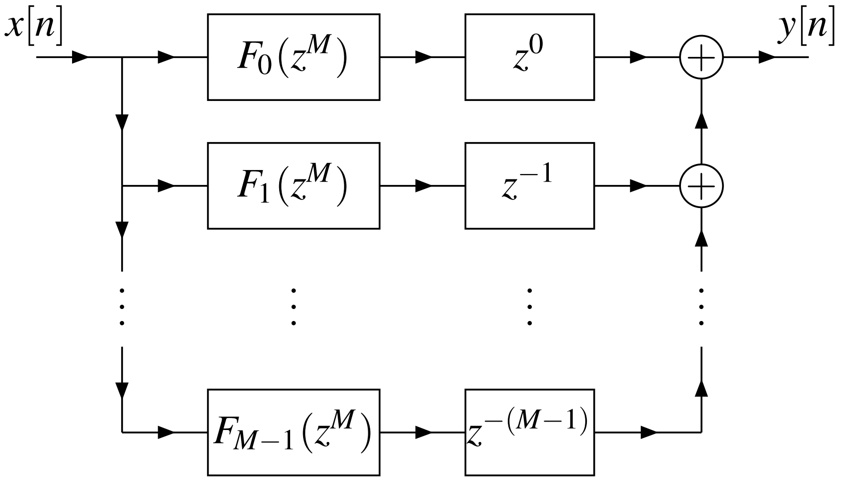


Conceptual

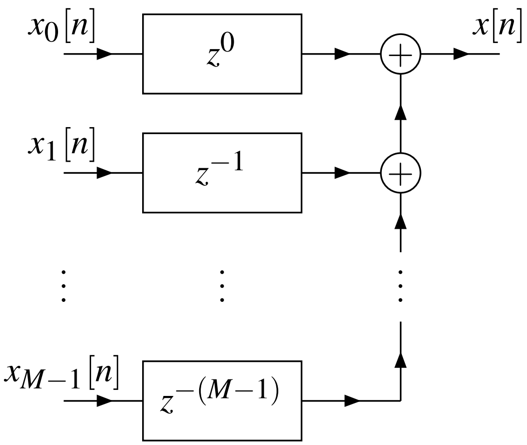


Practical

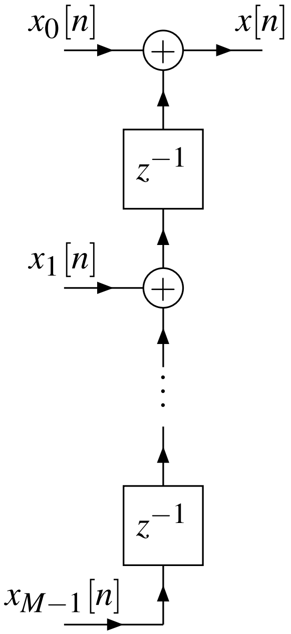
Type-1 Polyphase Representation and Advance/Delay Chains



Type-1 Polyphase Representation



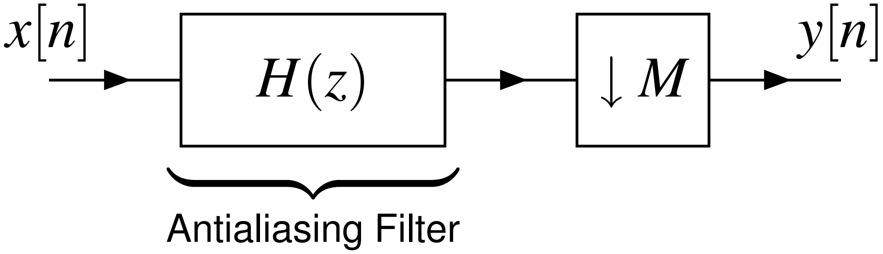
Conceptual



Practical

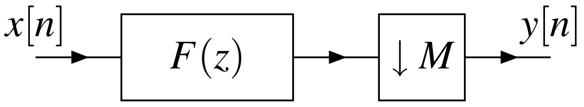
Efficient Decimation

- Earlier, the M-fold decimator was introduced, as shown below.

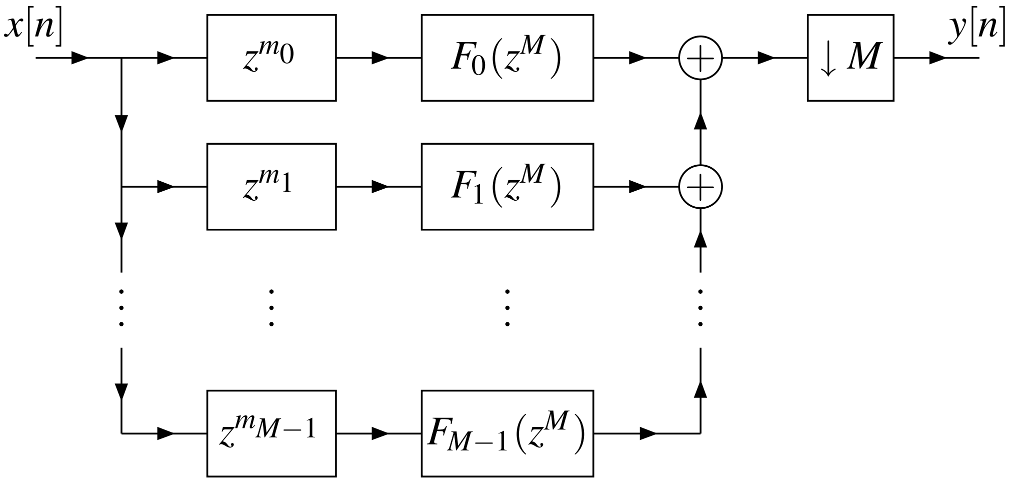


- In practice, however, a decimator would never be implemented directly using the above structure, since this structure is extremely computationally inefficient.
- Most of the samples computed by the convolution operation are discarded by the subsequent downsampling operation.
- For example, if $M = 2$, half of the results computed by convolution (i.e., filtering operation) are discarded, which is very inefficient. If $M > 2$, the inefficiency is even worse.
- By representing the antialiasing filter in M-phase polyphase form and using the noble identities, however, the above inefficiency can be eliminated.

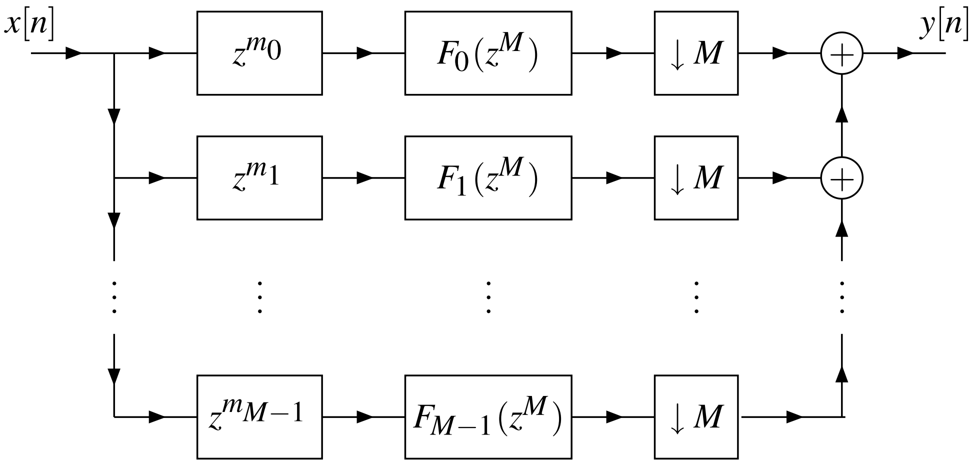
Efficient Decimation (Continued)



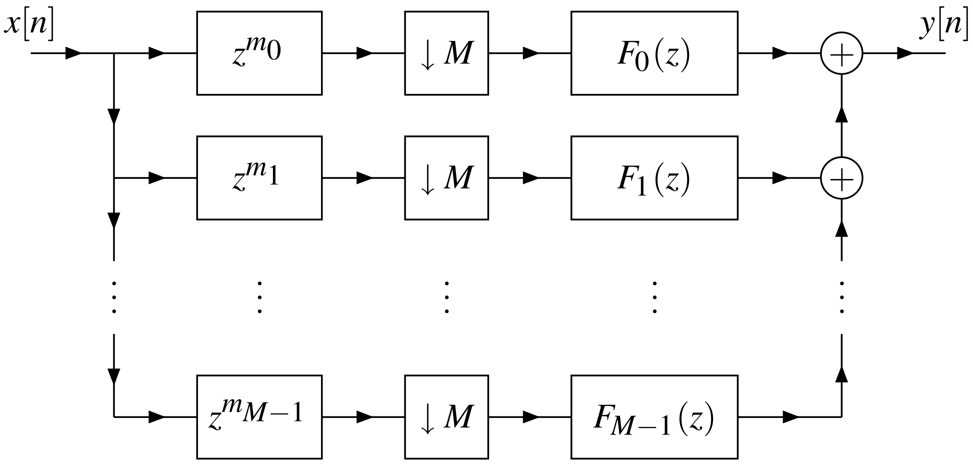
1) Original



2) After Implementing Filter in Polyphase Form



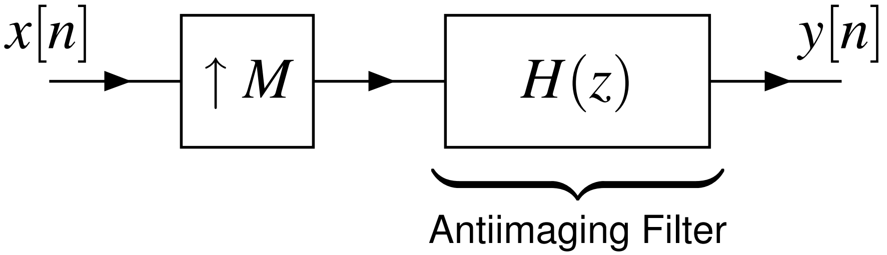
3) After Interchanging Downsamplers and Adders



4) After Interchanging Downsamplers and Filters

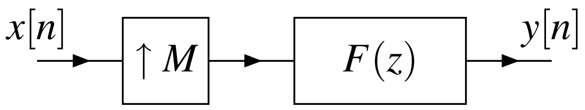
Efficient Interpolation

- Earlier, the M-fold interpolator was introduced, as shown below.

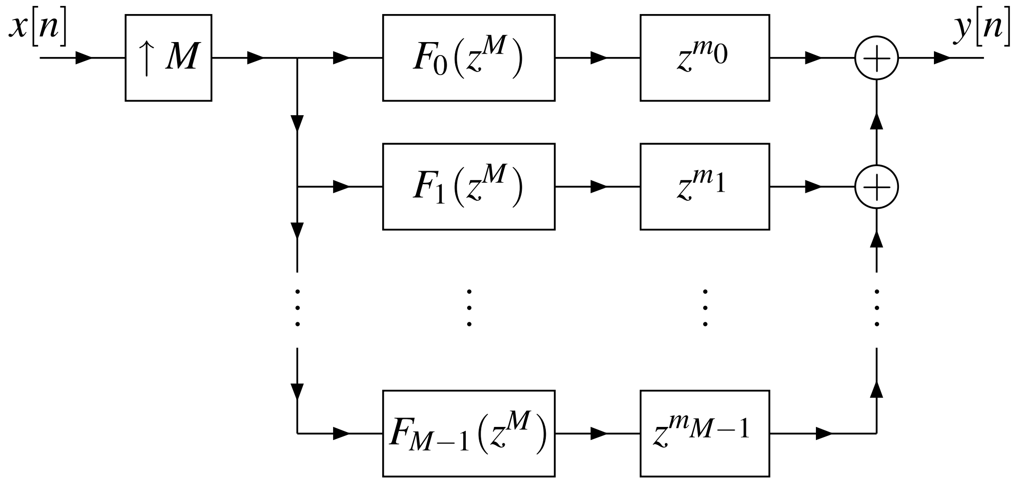


- In practice, however, an interpolator would never be implemented directly using the above structure, since this structure is extremely computationally inefficient.
- Due to the insertion of zeros by the upsampling operation, the subsequent convolution operation involves many multiplications by zero, resulting in most of the terms in the convolutional sum being zero.
- Repeated multiplication by zero and addition of zero is grossly inefficient.
- By representing the antiimaging filter in M-phase polyphase form and using the noble identities, however, this inefficiency can be eliminated.

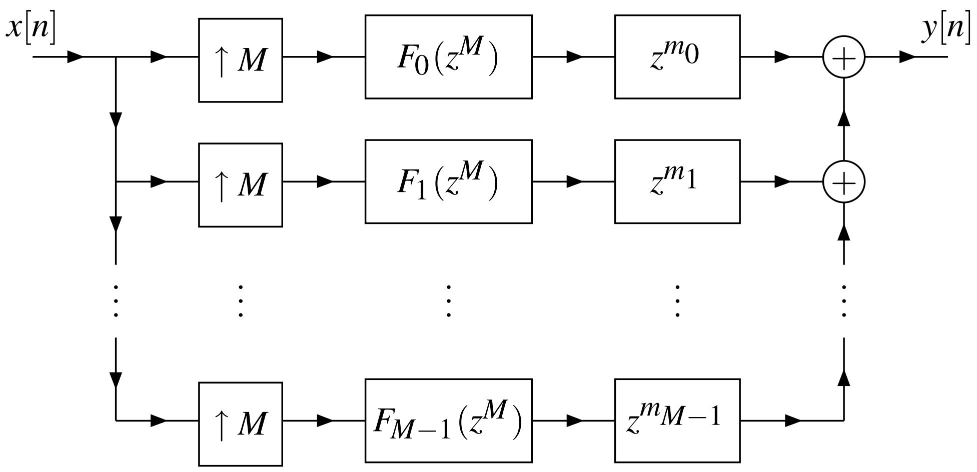
Efficient Interpolation (Continued)



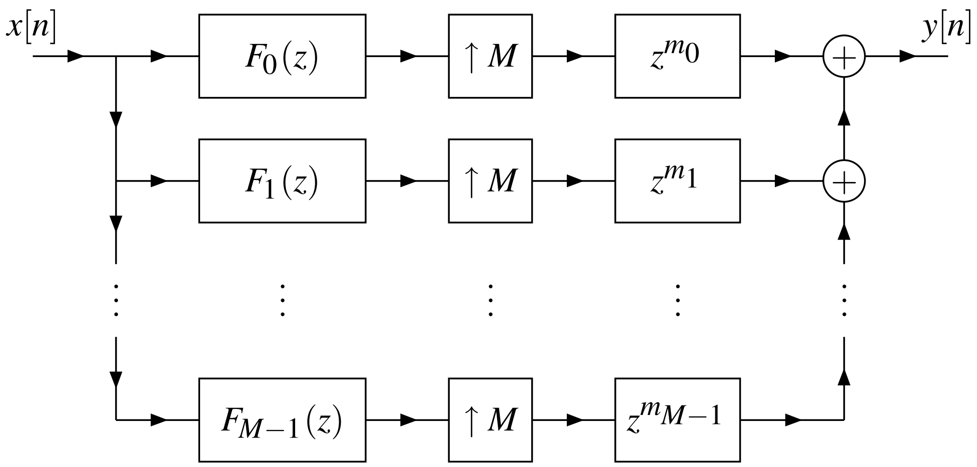
1) Original



2) After Implementing Filter in Polyphase Form



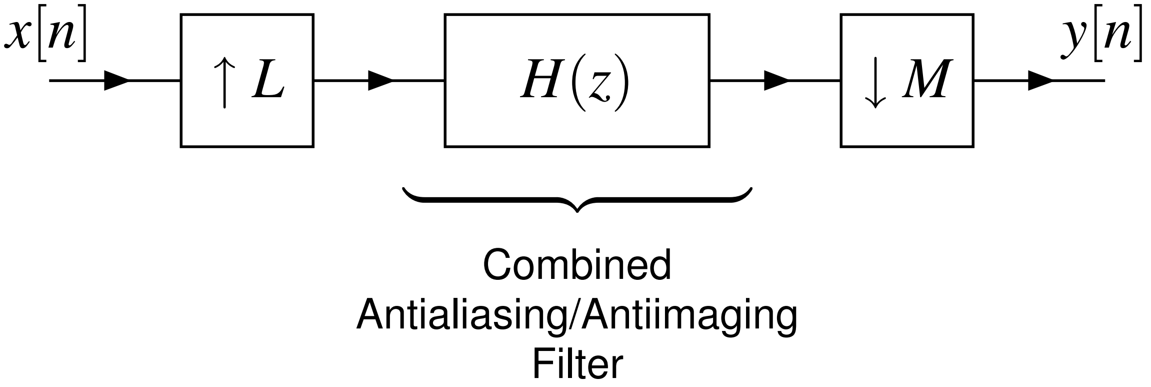
3) After Moving/Replicating Upsamplers



4) After Interchanging Upsamplers and Filters

Efficient Rational Sampling Rate Conversion

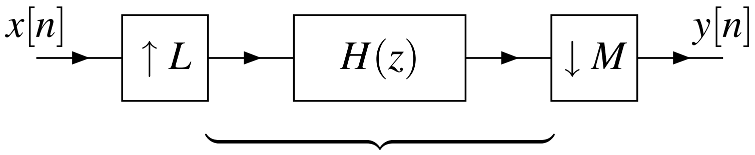
- Earlier, we consider rational sampling-rate converter as shown below.



Combined
Antialiasing/Antiimaging
Filter

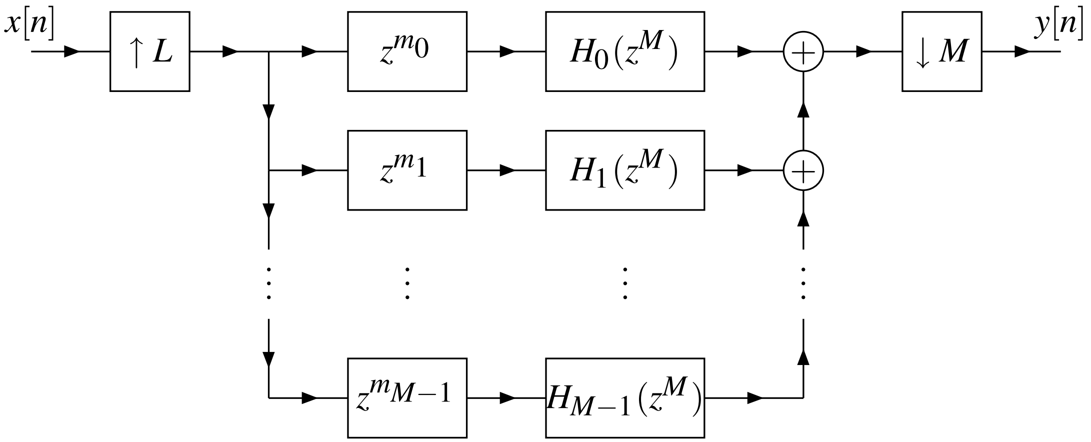
- In practice, a rational sampling-rate converter would never be implemented directly using the computational structure shown above, as this structure is grossly inefficient.
- By using polyphase techniques, a much more computationally efficient structure for a rational sampling rate converter can be obtained.

First Attempt: Upsampler Plus Efficient Decimator

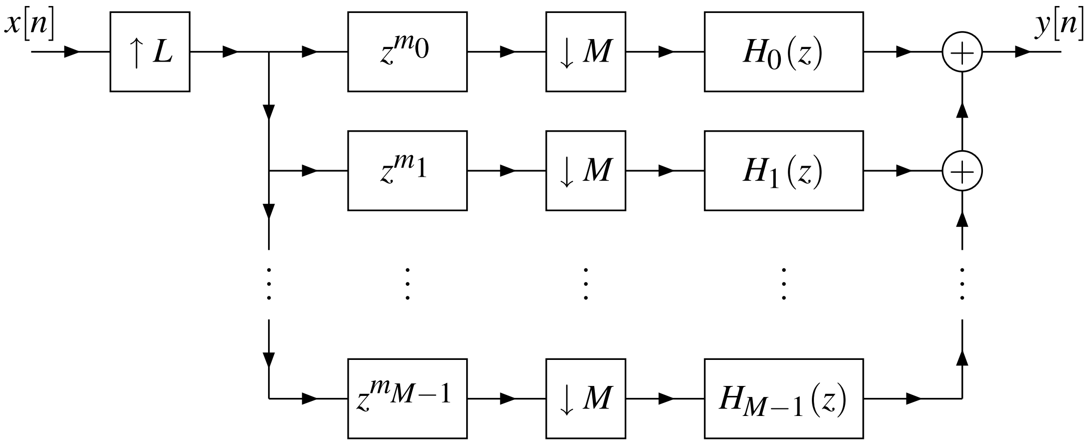


Combined
Antialiasing/Antiimaging
Filter

1) Original

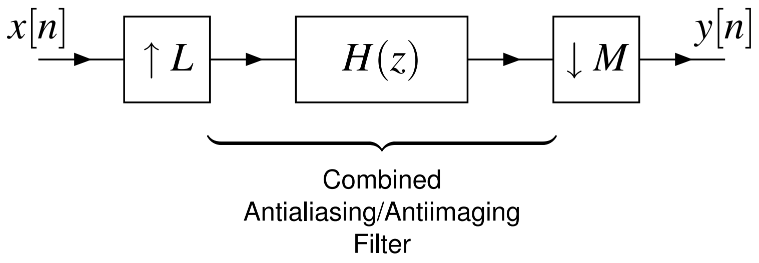


2) After Implementing Filter in Polyphase Form

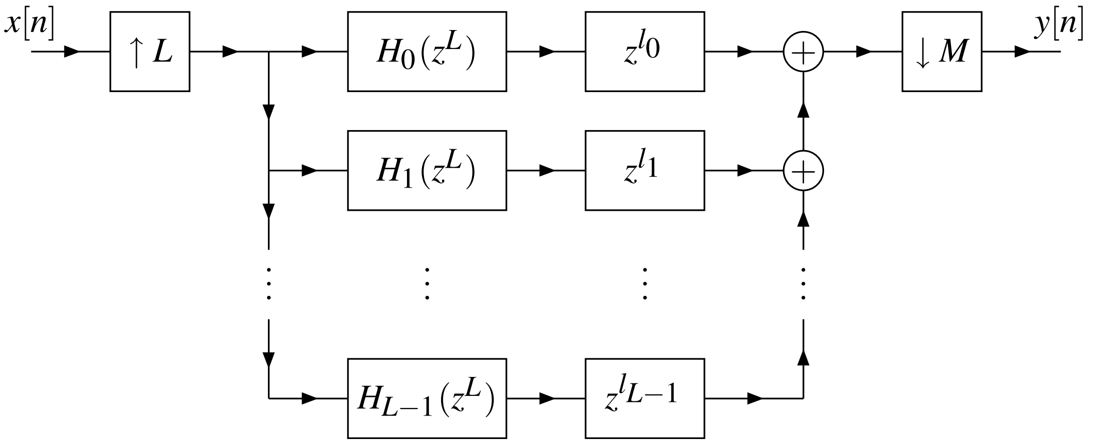


3) After Moving Filtering Across Downsamplers

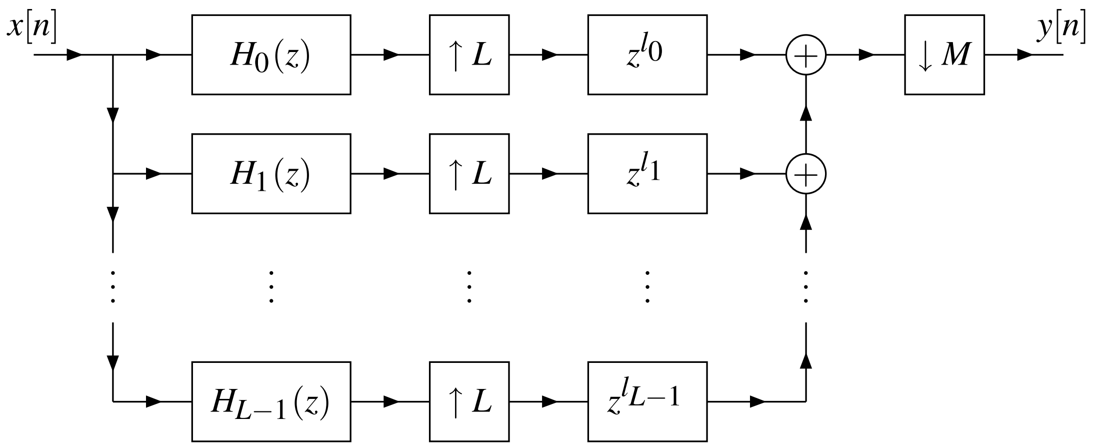
Second Attempt: Efficient Interpolator Plus Downsampler



1) Original



2) After Implementing Filter in Polyphase Form

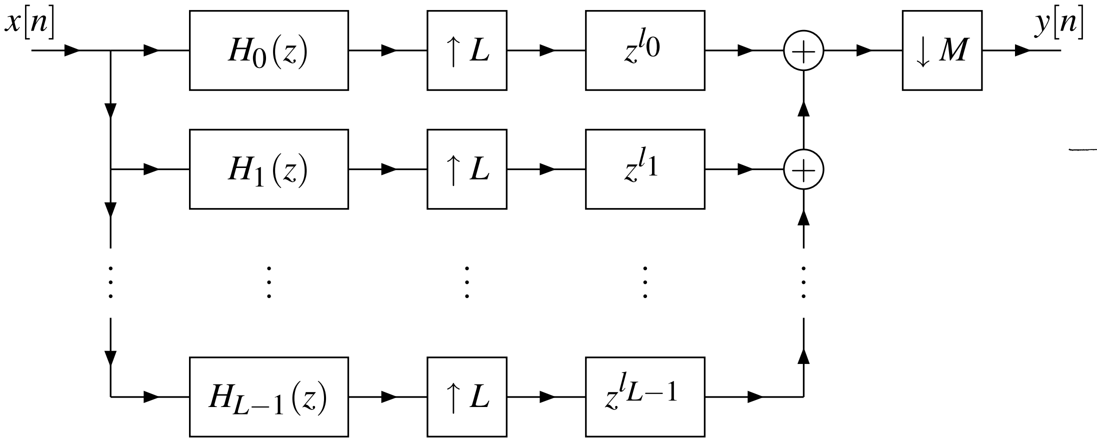


3) After Moving Filtering Across Upsamplers

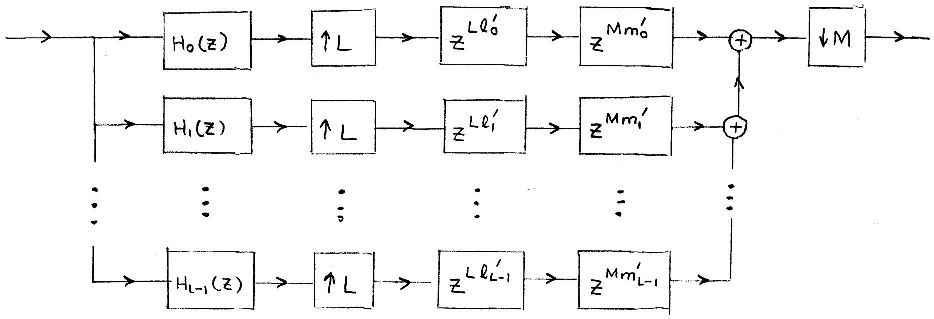
Efficient Conversion

- We can still do better in terms of efficiency.
- We start with the system obtained by moving the filtering operation across upsampling (as shown on previous slide).
- Without loss of generality, assume that L and M are coprime.
- By Bezout's identity, since L and M are coprime, for each l_k, there exist $l'_k, m'_k \in \mathbb{Z}$ such that $l_k = Ll'_k + Mm'_k$.
- Each delay/advance can be split into two.
- Further manipulation, allows a much more computationally efficient structure to be obtained.
- In the final system, all filtering is performed at the lowest possible sampling rate (i.e., after downsampling and before upsampling).

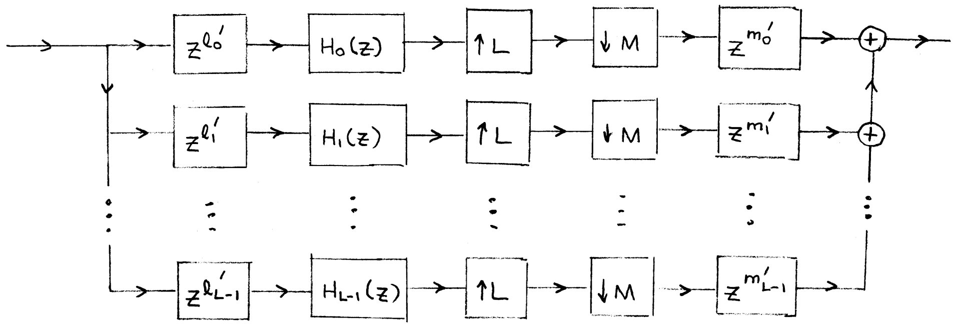
Efficient Conversion (Continued)



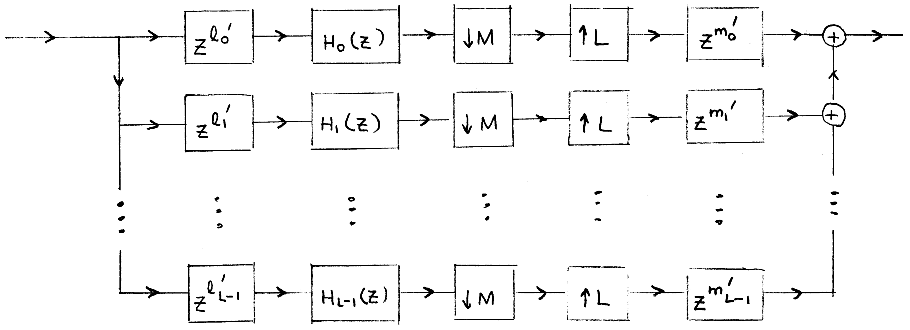
1) Efficient Interpolator Plus Downsampler



2) After Splitting Delays ($z^{l_k} = z^{Ll_k'} z^{Mm_k'}$)

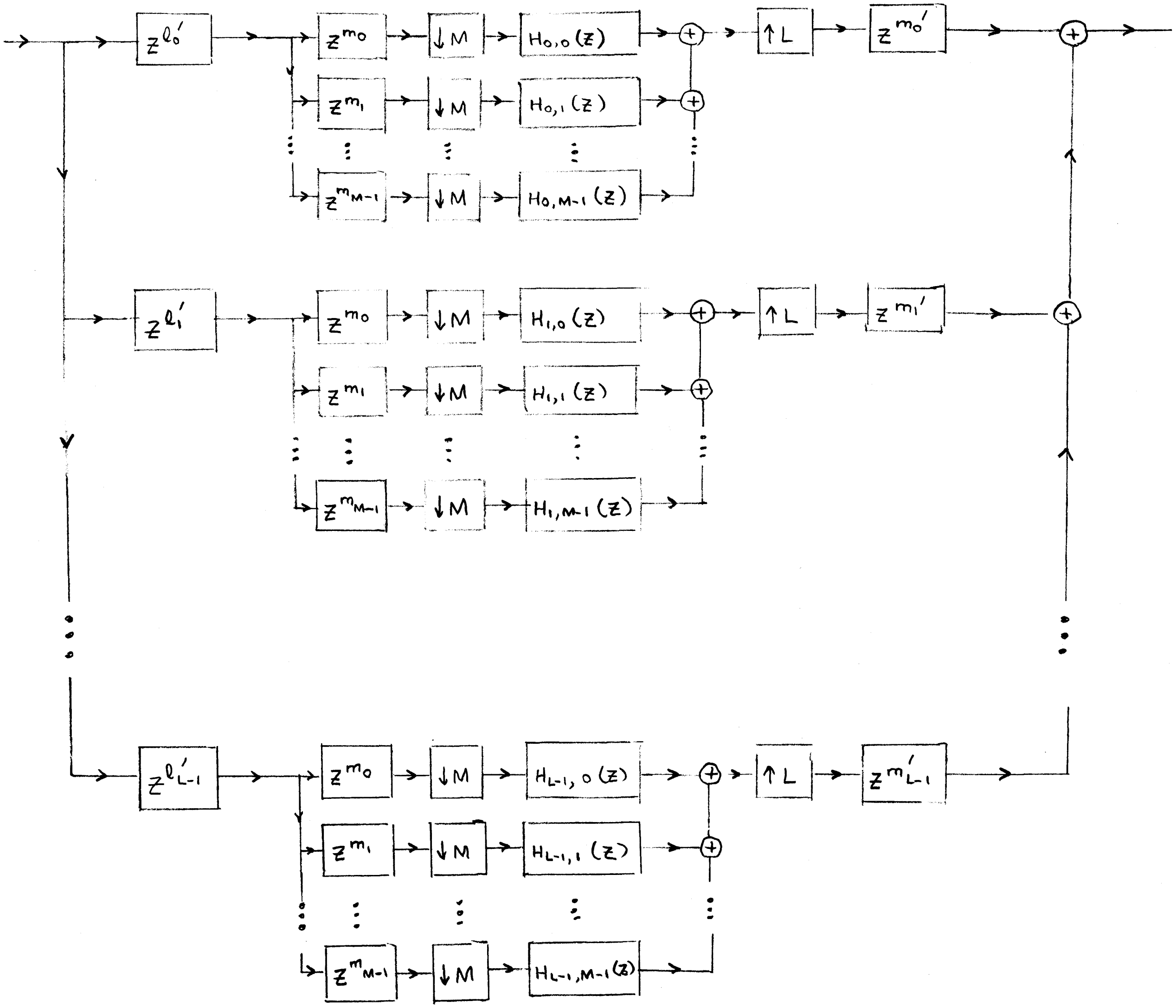


3) After Delays Moved Across Downsamplers/Upsamplers



4) After Downsamplers/Upsamplers Interchanged (L and M coprime)

Efficient Conversion (Continued 2)



5) After Polyphase Filtering and Downsampling Interchanged

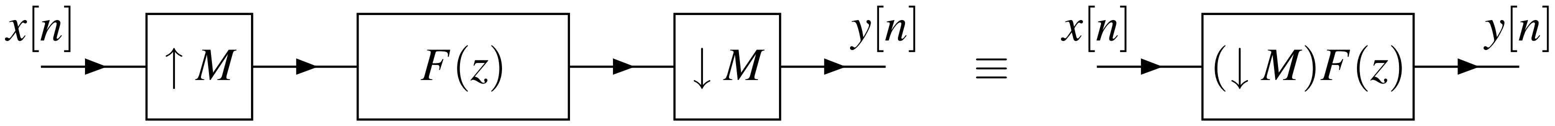
Polyphase Identity

- We sometimes encounter an M-fold upsampling operation followed by an M-fold downsampling operation with filtering in between.
- One useful identity in relation to such situations is given by the result below.
- **Polyphase identity.** Let F and X denote $\mathcal{Z}$ transforms. Then, we have

$$(\downarrow M)\left(F(z)\left[(\uparrow M)X(z)\right]\right) = F_0(z)X(z),$$

where $F_0(z) = (\downarrow M)F(z)$.

- The above relationship has the interpretation shown below.

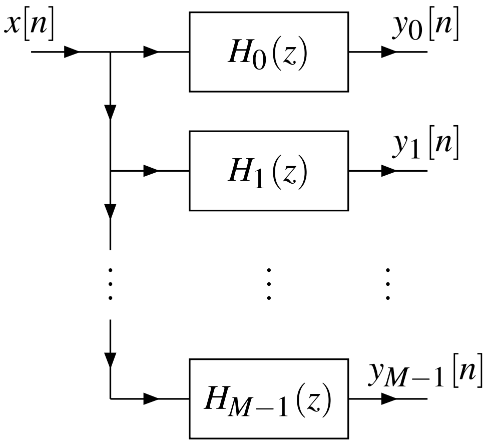


Section 3.2

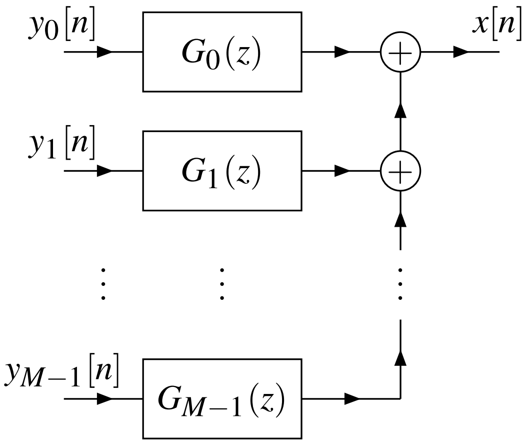
Multirate Filter Banks

Filter Banks

- A collection of filters having either a common input or common output is called a **filter bank**.
- When the filters share a common input, they form an **analysis bank**. The filters of an analysis bank are called **analysis filters**.
- When the filters share a common output, they form a **synthesis bank**. The filters of a synthesis bank are called **synthesis filters**.
- The outputs of an analysis bank and inputs of a synthesis bank are referred to as **subband signals**.
- The frequency responses of the filters may be non-overlapping, marginally overlapping, or greatly overlapping, depending on the application.



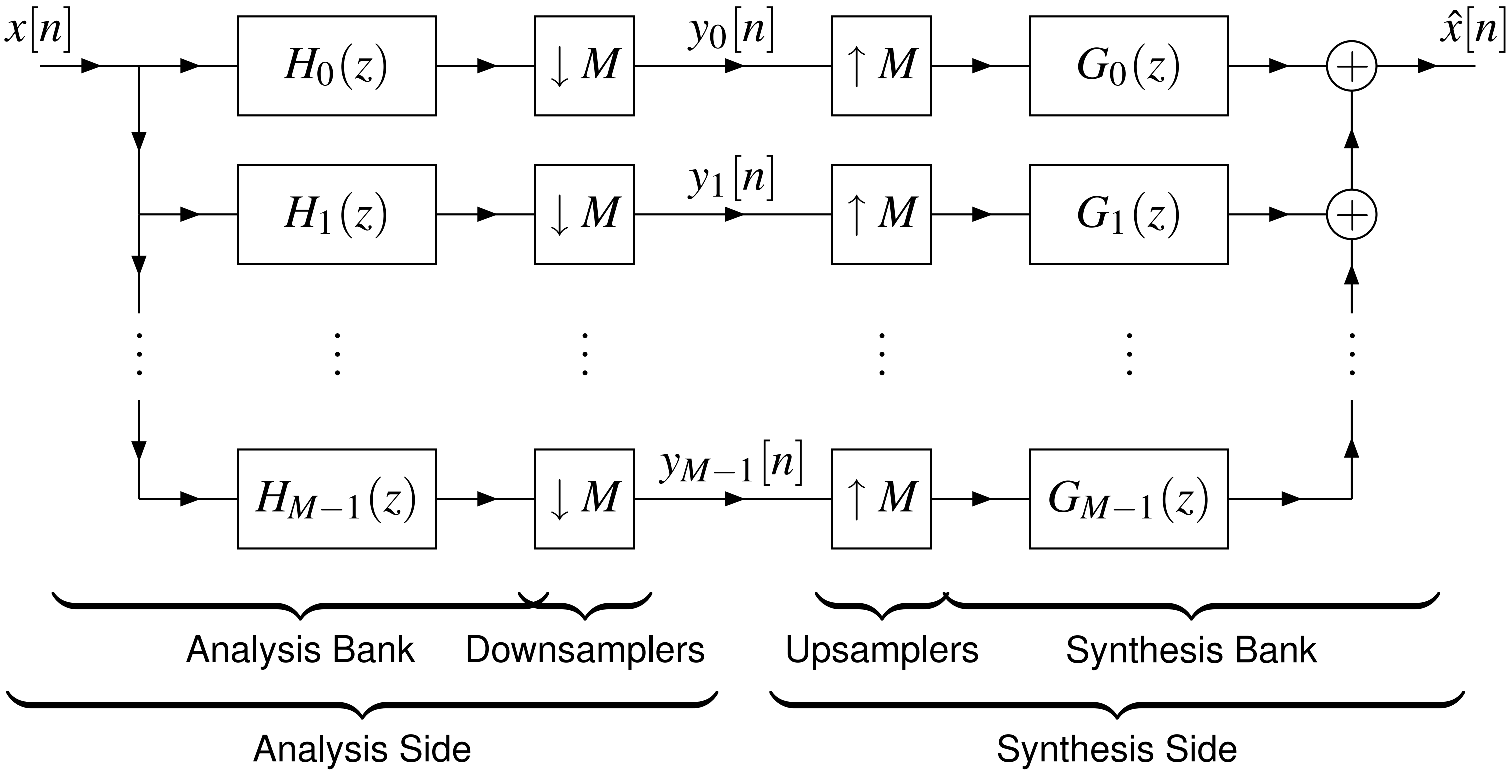
Analysis Bank



Synthesis Bank

Uniformly Maximally Decimated (UMD) Filter Banks

- Although many filter bank configurations exist, an extremely useful one is the M-channel **uniformly maximally decimated (UMD) filter bank**, which has the general structure shown below.

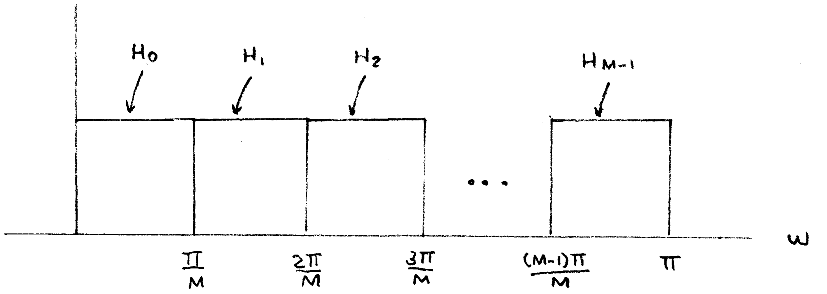


Alias-Free and Perfect-Reconstruction Properties

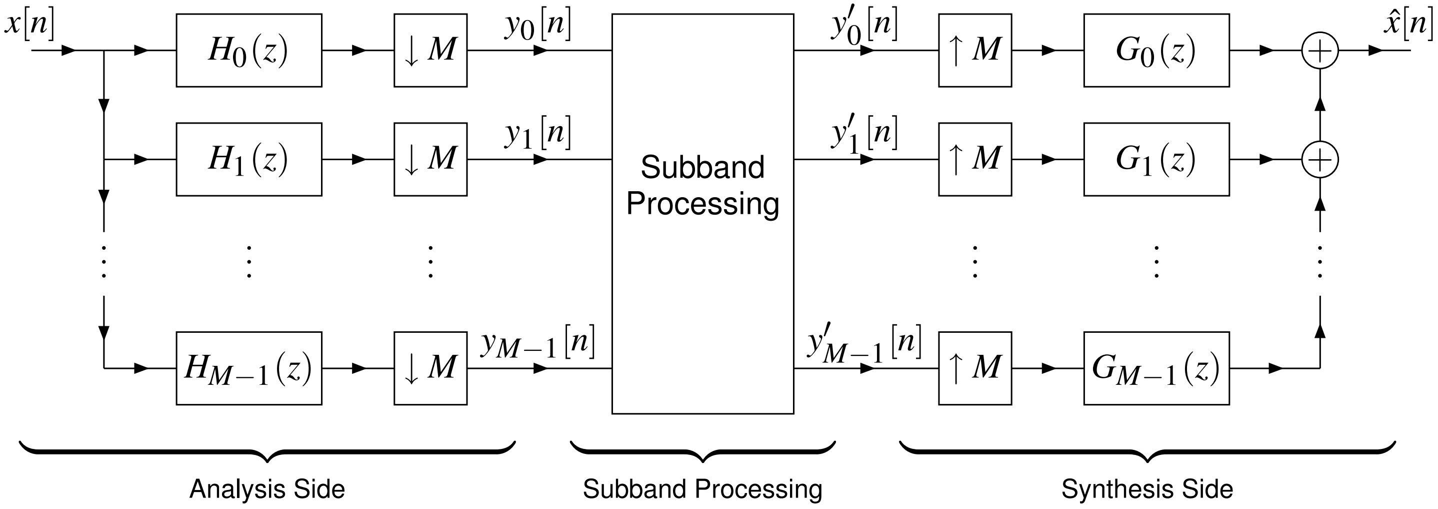
- Let us consider an M-channel UMD filter bank as a system with the ***input*** x and ***output*** $\hat{x}$.
- In the most general case, the system is linear and ***periodically time varying*** (with period M).
- Due to aliasing (and imaging), the system may not be time invariant. In practice, at least some limited amount of aliasing and imaging always occurs, due to the use of ***nonideal filters***.
- Often, it is possible to choose the analysis/synthesis filters such that aliasing is completely cancelled.
- If all of the aliasing components completely cancel, the system is time invariant and is said to be **alias free**.
- If, for all sequences x and some integer n_0, the system is such that
$$\hat{x}[n] = x[n-n_0] \quad \text{for all } n \in \mathbb{Z},$$
the system is said to have the **perfect reconstruction (PR)** property.
- In the special case that $n_0 = 0$, the system is said to have the **shift-free PR** property.
- UMD filter banks with the shift-free PR property play an important role in the context of ***wavelet systems***.

Utility of UMD Filter Banks

- M-channel filter bank can be used to partition signal into M frequency bands (which are determined by analysis filters $\{H_k\}_{k=0}^{M-1}$) as shown below



- therefore, filter bank potentially useful when desirable to process signal in terms of its different frequency bands
- in practice, often have processing block inserted between analysis and synthesis sides of filter bank like shown below



Utility of UMD Filter Banks (Continued)

- often PR system desired so that any difference between x and $\hat{x}$ is due to subband processing and not distortion caused by filter bank itself
- also basic building block in computational structure for discrete wavelet transform (DWT)
- examine several applications later

Time-Domain Characterization of UMD Filter Banks

- In the time-domain, the ***analysis side*** of the filter bank is characterized by the equation

$$y_k[n] = \sum_{l \in \mathbb{Z}} x[l] h_k[Mn - l].$$

- Similarly, the ***synthesis side*** of the filter bank is characterized by the equation

$$\hat{x}[n] = \sum_{k=0}^{M-1} \sum_{l \in \mathbb{Z}} y_k[l] g_k[n - Ml].$$

- The analysis and synthesis sides ***together*** can be characterized by combining the above equations.

$\mathcal{Z}$-Domain Characterization of UMD Filter Banks

- In the $\mathcal{Z}$-domain, the ***analysis*** and ***synthesis*** sides of the filter bank are characterized, respectively, by the equations

$$Y_k(z) = \frac{1}{M}\sum_{l=0}^{M-1} H_k(z^{1/M}W^l)X(z^{1/M}W^l) \quad \text{and} \quad \hat{X}(z) = \sum_{k=0}^{M-1} G_k(z)Y_k(z^M).$$

- The analysis and synthesis sides of the filter bank ***together*** are characterized by the equation

$$\hat{X}(z) = \sum_{l=0}^{M-1} A_l(z)X(zW^l) \quad \text{where} \quad A_l(z) = \frac{1}{M}\sum_{k=0}^{M-1} G_k(z)H_k(zW^l).$$

- We have: $\hat{X}(z) = A_0(z)X(z) + \underbrace{A_1(z)X(zW) + \ldots + A_{M-1}(z)X(zW^{M-1})}_{\textit{aliasing terms}}.$
- The system is ***alias free*** with ***transfer function $A_0(z)$*** if and only if $A_l(z) \equiv 0$ for $l \in \{1, 2, \ldots, M-1\}$.
- The system has the ***PR property*** if and only if it is alias free and $A_0(z)$ is of the form $A_0(z) = z^{-n_0}$ where $n_0 \in \mathbb{Z}$.

Modulation Matrices

- The **analysis modulation matrix** $\boldsymbol{H}_{\mathrm{m}}(z)$ of the filter bank is defined as

$$\boldsymbol{H}_{\mathrm{m}}(z) = \begin{bmatrix} H_0(z) & H_0(zW) & \cdots & H_0(zW^{M-1}) \\ H_1(z) & H_1(zW) & \cdots & H_1(zW^{M-1}) \\ \vdots & \vdots & \ddots & \vdots \\ H_{M-1}(z) & H_{M-1}(zW) & \cdots & H_{M-1}(zW^{M-1}) \end{bmatrix}.$$

- The **synthesis modulation matrix** $\boldsymbol{G}_{\mathrm{m}}(z)$ is defined as

$$\boldsymbol{G}_{\mathrm{m}}(z) = \begin{bmatrix} G_0(z) & G_1(z) & \cdots & G_{M-1}(z) \\ G_0(zW) & G_1(zW) & \cdots & G_{M-1}(zW) \\ \vdots & \vdots & \ddots & \vdots \\ G_0(zW^{M-1}) & G_1(zW^{M-1}) & \cdots & G_{M-1}(zW^{M-1}) \end{bmatrix}.$$

- Note: In the case of $\boldsymbol{H}_{\mathrm{m}}(z)$, $H_k(z)$ and its aliased versions appear in the kth *row* of $\boldsymbol{H}_{\mathrm{m}}(z)$, whereas in the case of $\boldsymbol{G}_{\mathrm{m}}(z)$, $G_k(z)$ and its aliased versions appear in the kth *column* of $\boldsymbol{G}_{\mathrm{m}}(z)$.

Modulation-Domain Representation of UMD Filter Banks

- The earlier equation characterizing the filter bank in the $\mathcal{Z}$-domain can be written in matrix form as

$$\hat{X}(z) = \begin{bmatrix} X(z) & X(zW) & \cdots & X(zW^{M-1}) \end{bmatrix} \boldsymbol{A}(z),$$

where $W = W_M = e^{-j2\pi/M}$,

$$\boldsymbol{A}(z) = \begin{bmatrix} A_0(z) \\ A_1(z) \\ \vdots \\ A_{M-1}(z) \end{bmatrix} = \tfrac{1}{M}\boldsymbol{H}_{\mathrm{m}}^T(z)\boldsymbol{g}(z), \quad \text{and} \quad \boldsymbol{g}(z) = \begin{bmatrix} G_0(z) \\ G_1(z) \\ \vdots \\ G_{M-1}(z) \end{bmatrix}.$$

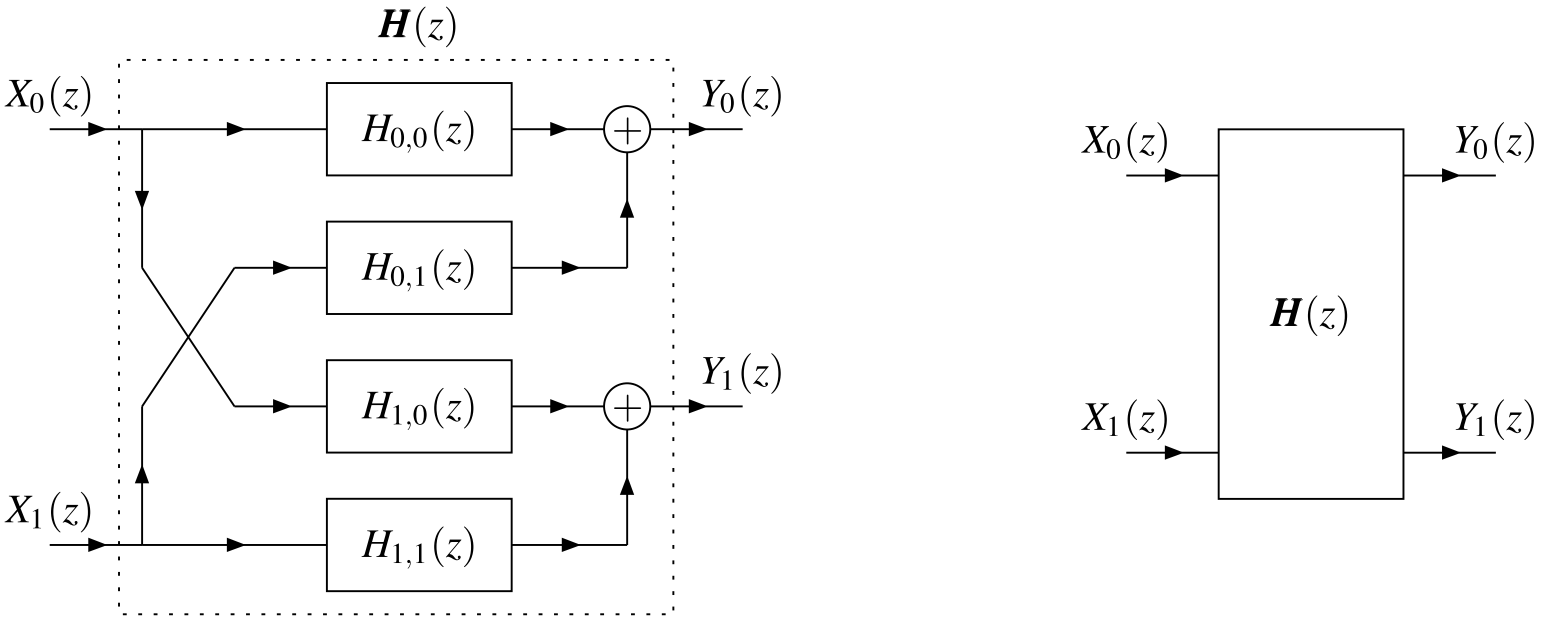
- Suppose that, given a set of analysis filters, we want to find the set of synthesis filters that results in a particular alias free or PR system.
- We can solve for $\boldsymbol{g}(z)$ in terms of $\boldsymbol{H}_{\mathrm{m}}(z)$ and $\boldsymbol{A}(z)$ to obtain

$$\boldsymbol{g}(z) = M\boldsymbol{H}_{\mathrm{m}}^{-T}(z)\boldsymbol{A}(z).$$

- FIR analysis filters but IIR synthesis filters? stability of IIR filters? synthesis filters of much higher order than analysis filters?

Transfer Matrices for LTI MIMO Systems

- Any M-input M-output LTI system can be completely characterized by an $M \times M$ matrix of transfer functions, called a **transfer matrix**.
- Consider the case of $M = 2$ below.



- We have:

$$\begin{bmatrix} Y_0(z) \\ Y_1(z) \end{bmatrix} = \underbrace{\begin{bmatrix} H_{0,0}(z) & H_{0,1}(z) \\ H_{1,0}(z) & H_{1,1}(z) \end{bmatrix}}_{\boldsymbol{H}(z)} \begin{bmatrix} X_0(z) \\ X_1(z) \end{bmatrix} = \begin{bmatrix} H_{0,0}(z)X_0(z) + H_{0,1}(z)X_1(z) \\ H_{1,0}(z)X_0(z) + H_{1,1}(z)X_1(z) \end{bmatrix}.$$

- Thus, the system is completely characterized by the transfer matrix $\boldsymbol{H}(z)$.

Polyphase Representation of UMD Filter Bank

- We can apply polyphase methods to UMD filter banks, in order to obtain the so called polyphase representation of a filter bank.
- Basically, we represent the filters of ***analysis and synthesis banks in polyphase form***.
- The polyphase representation of filter banks is of ***great importance***.
- It ***greatly simplifies*** the study of filter banks.
- It suggests an ***efficient means for implementing*** filter banks.

Polyphase Representation of Analysis Side

- We can express the transfer functions $\{H_k\}_{k=0}^{M-1}$ of the ***analysis filters in polyphase form*** as
$$H_k(z) = \sum_{p=0}^{M-1} z^{m_p} H_{k,p}(z^M).$$
- Note that the ***same type*** of polyphase representation is used for each analysis filter (i.e., $\{m_k\}_{k=0}^{M-1}$ is fixed for all analysis filters).
- The resulting set of equations can be rewritten ***in matrix form*** as
$$\underbrace{\begin{bmatrix} H_0(z) \\ H_1(z) \\ \vdots \\ H_{M-1}(z) \end{bmatrix}}_{\boldsymbol{h}(z)} = \underbrace{\begin{bmatrix} H_{0,0}(z^M) & H_{0,1}(z^M) & \cdots & H_{0,M-1}(z^M) \\ H_{1,0}(z^M) & H_{1,1}(z^M) & \cdots & H_{1,M-1}(z^M) \\ \vdots & \vdots & \ddots & \vdots \\ H_{M-1,0}(z^M) & H_{M-1,1}(z^M) & \cdots & H_{M-1,M-1}(z^M) \end{bmatrix}}_{\boldsymbol{H}_\mathrm{p}(z^M)} \underbrace{\begin{bmatrix} z^{m_0} \\ z^{m_1} \\ \vdots \\ z^{m_{M-1}} \end{bmatrix}}_{\boldsymbol{v}(z)}$$
or more compactly as
$$\boldsymbol{h}(z) = \boldsymbol{H}_\mathrm{p}(z^M)\boldsymbol{v}(z).$$
- The preceding equation ***completely characterizes*** the analysis side.
- The quantity $\boldsymbol{H}_\mathrm{p}(z)$ is referred to as the **analysis polyphase matrix**.

Polyphase Representation of Synthesis Side

- We can express the transfer functions $\{G_k\}_{k=0}^{M-1}$ of the ***synthesis filters in polyphase form*** as

$$G_k(z) = \sum_{p=0}^{M-1} z^{l_p} G_{p,k}(z^M).$$

- Note that the ***same type*** of polyphase representation is used for each synthesis filter (i.e., $\{l_k\}_{k=0}^{M-1}$ is fixed for all synthesis filters).
- The above equation expressed ***in matrix form*** becomes

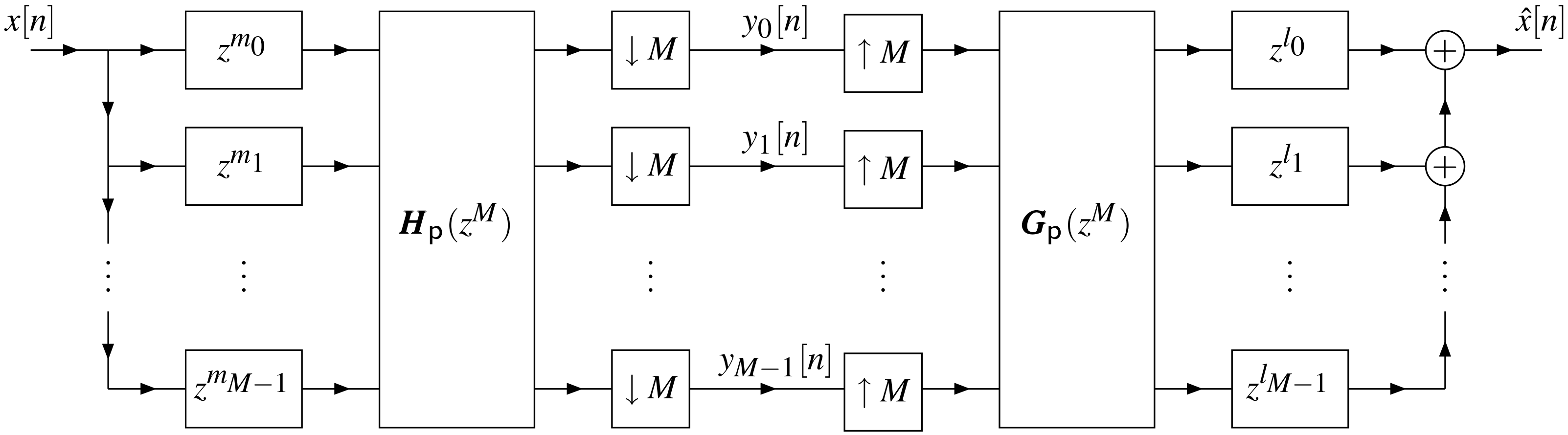
$$\underbrace{\begin{bmatrix} G_0(z) & G_1(z) & \cdots & G_{M-1}(z) \end{bmatrix}}_{\boldsymbol{g}^T(z)} = \underbrace{\begin{bmatrix} z^{l_0} & z^{l_1} & \ldots & z^{l_{M-1}} \end{bmatrix}}_{\boldsymbol{u}^T(z)} \underbrace{\begin{bmatrix} G_{0,0}(z^M) & G_{0,1}(z^M) & \cdots & G_{0,M-1}(z^M) \\ G_{1,0}(z^M) & G_{1,1}(z^M) & \cdots & G_{1,M-1}(z^M) \\ \vdots & \vdots & \ddots & \vdots \\ G_{M-1,0}(z^M) & G_{M-1,1}(z^M) & \cdots & G_{M-1,M-1}(z^M) \end{bmatrix}}_{\boldsymbol{G}_{\mathrm{p}}(z^M)}$$

which can be written more concisely as

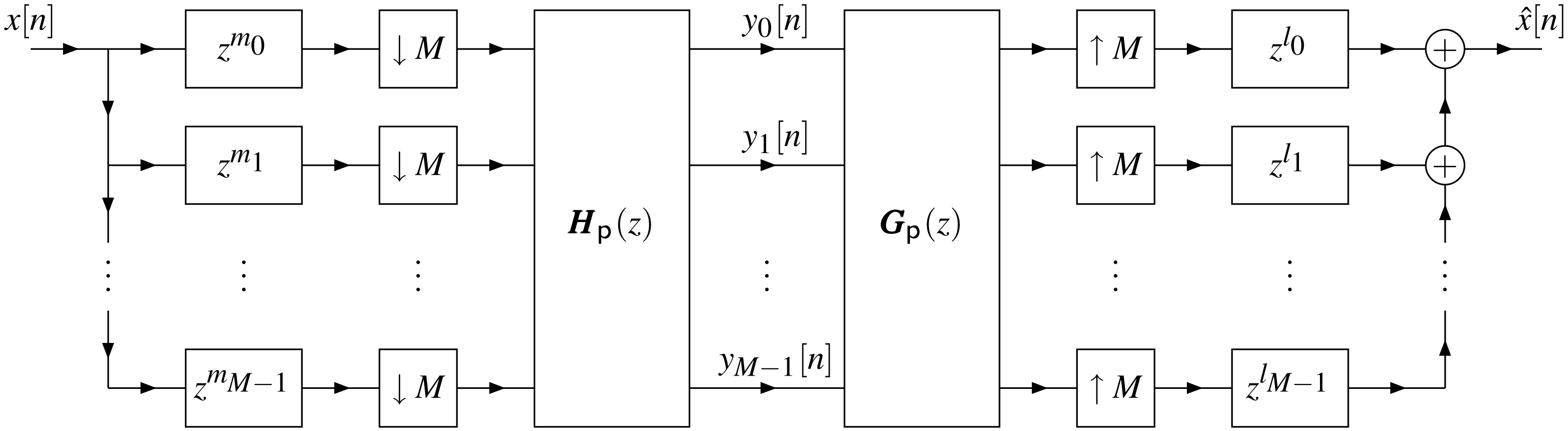
$$\boldsymbol{g}^T(z) = \boldsymbol{u}^T(z)\boldsymbol{G}_{\mathrm{p}}(z^M).$$

- The preceding equation ***completely characterizes*** the synthesis side.
- The quantity $\boldsymbol{G}_{\mathrm{p}}(z)$ is referred to as the **synthesis polyphase matrix**.

Polyphase Representation of UMD Filter Bank



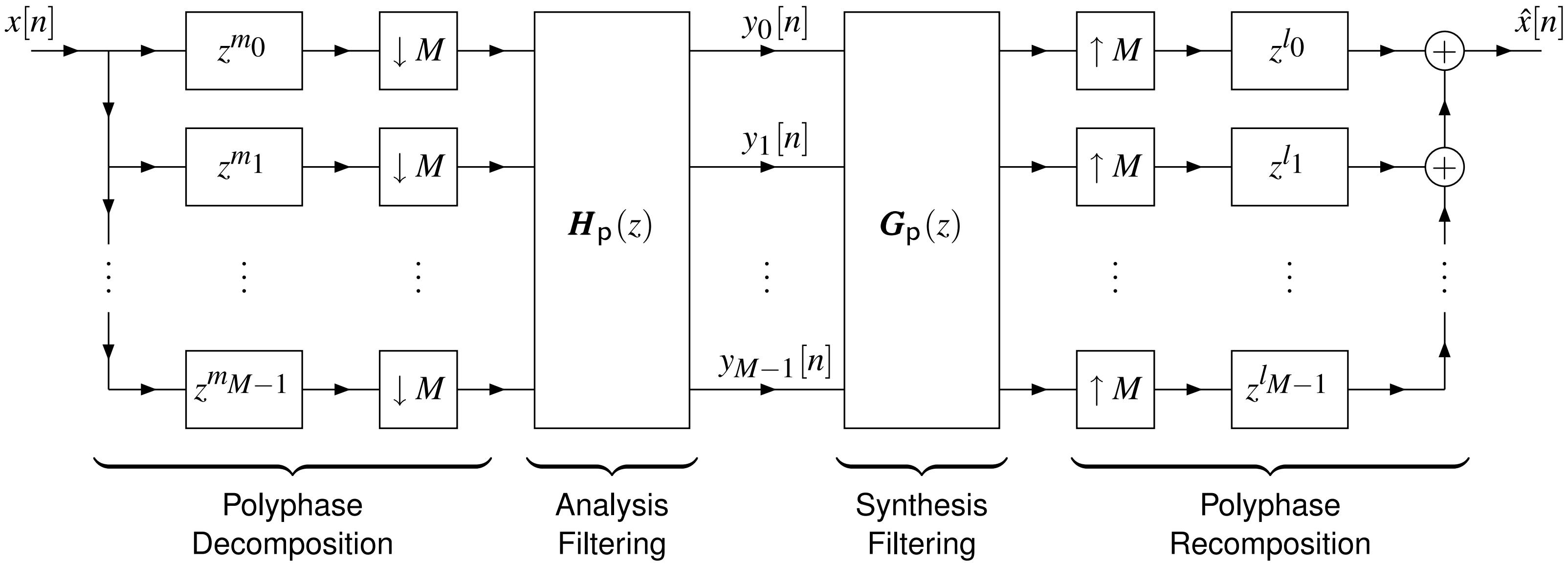
Before simplification with the noble identities



Polyphase representation

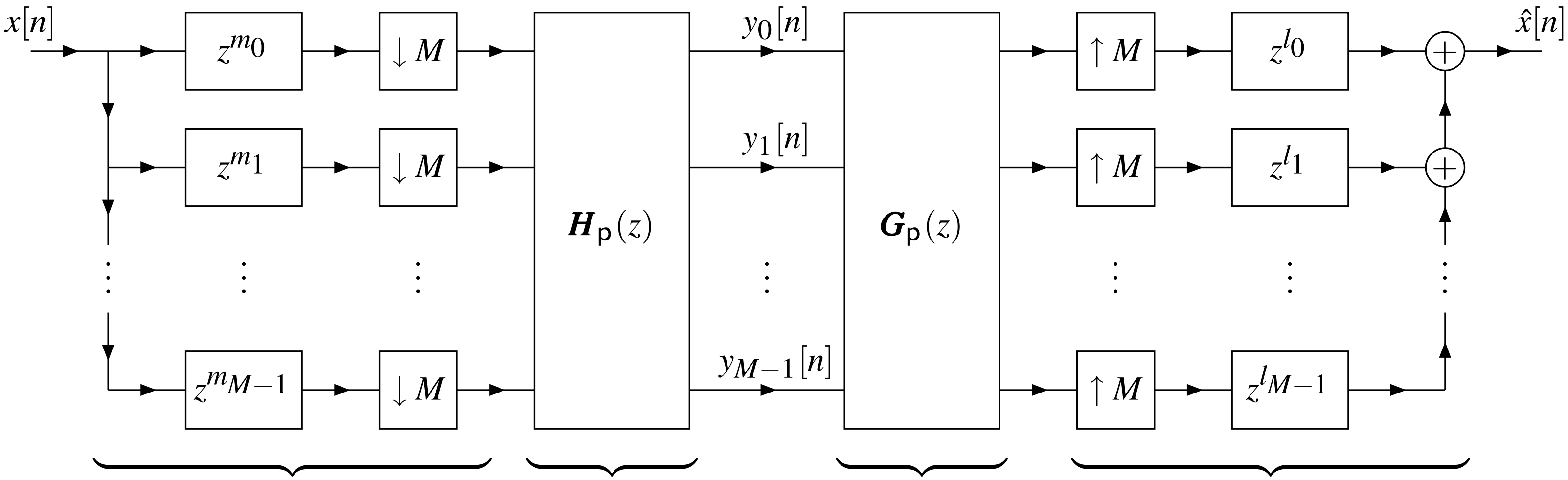
Polyphase Representation of UMD Filter Bank (Continued)

- With the polyphase representation of a UMD filter bank, the input x is split into its polyphase components (i.e., ***polyphase decomposition***).
- Then, the M polyphase components undergo analysis and synthesis filtering via ***M-input M-output LTI networks***.
- Lastly, the M polyphase components of the result are recombined to form the output $\hat{x}$ (i.e., ***polyphase recomposition***).



- The polyphase representation ***transforms*** a filter bank from a ***SISO linear periodically-time-varying*** system to a ***MIMO LTI*** system operating on polyphase components of signals.

Practical Utility of the Polyphase Representation



Polyphase Decomposition

Analysis Filtering

Synthesis Filtering

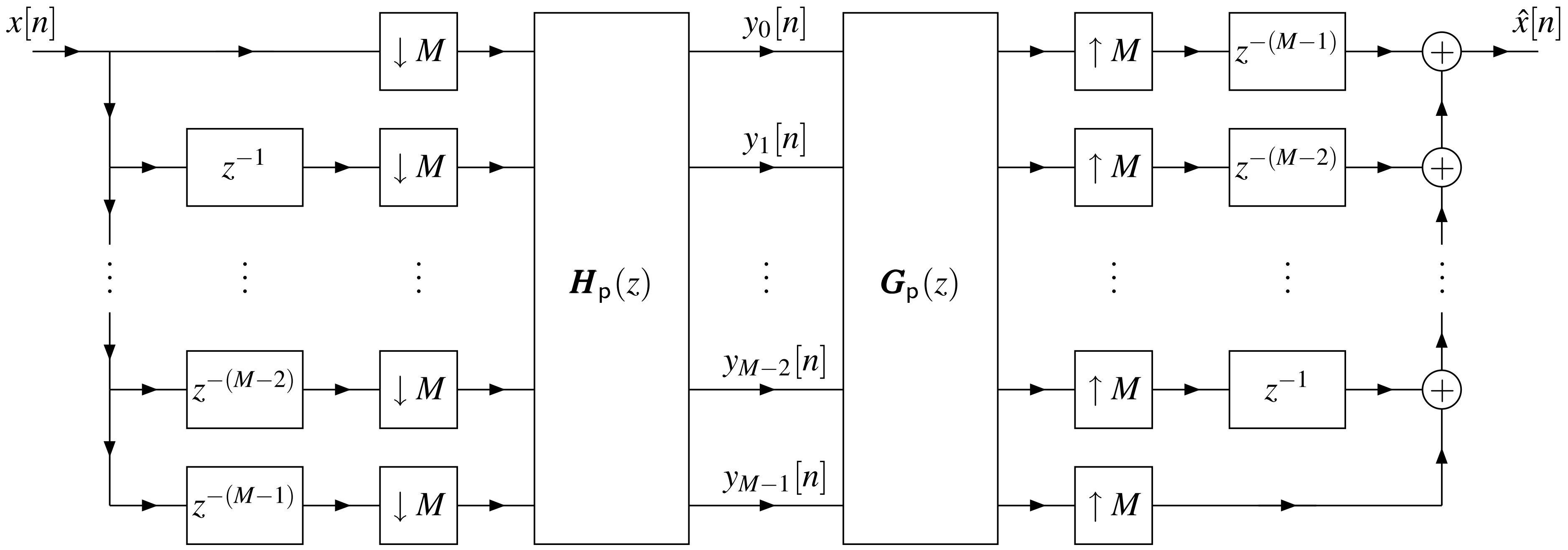
Polyphase Recomposition

- With the polyphase representation of a filter bank, all analysis/synthesis filtering is performed on the side of the downsamplers/upsamplers with the *lower* sampling density.
- Consequently, the polyphase representation provides a ***particularly efficient*** implementation strategy for filter banks.

Types of Polyphase Representations

- ***Considerable freedom*** exists in the choice of the parameters $\{m_k\}_{k=0}^{M-1}$ and $\{l_k\}_{k=0}^{M-1}$ used to form polyphase representations of the analysis and synthesis banks.
- For this reason, ***many variations*** on the polyphase representation of a UMD filter bank are possible.
- A polyphase representation that employs type-a and type-b polyphase representations for the analysis and synthesis sides of the filter bank, respectively, is denoted as **type-(a, b)**.
- In practice, the two ***most commonly used*** types of representation are type-(1,2) and type-(3,1).
- The type-(1,2) representation only employs unit delays for polyphase decomposition/recomposition, while the type-(3,1) representation uses ***unit advances*** as well as delays. Thus, the latter representation may not be suitable for use in some applications, due to ***causality constraints***.
- The type-(3,1) representation has nicer mathematical properties, since polyphase decomposition and recomposition are mathematical ***inverses***.
- The type-(3,1) representation is more directly relevant to ***wavelet systems***.

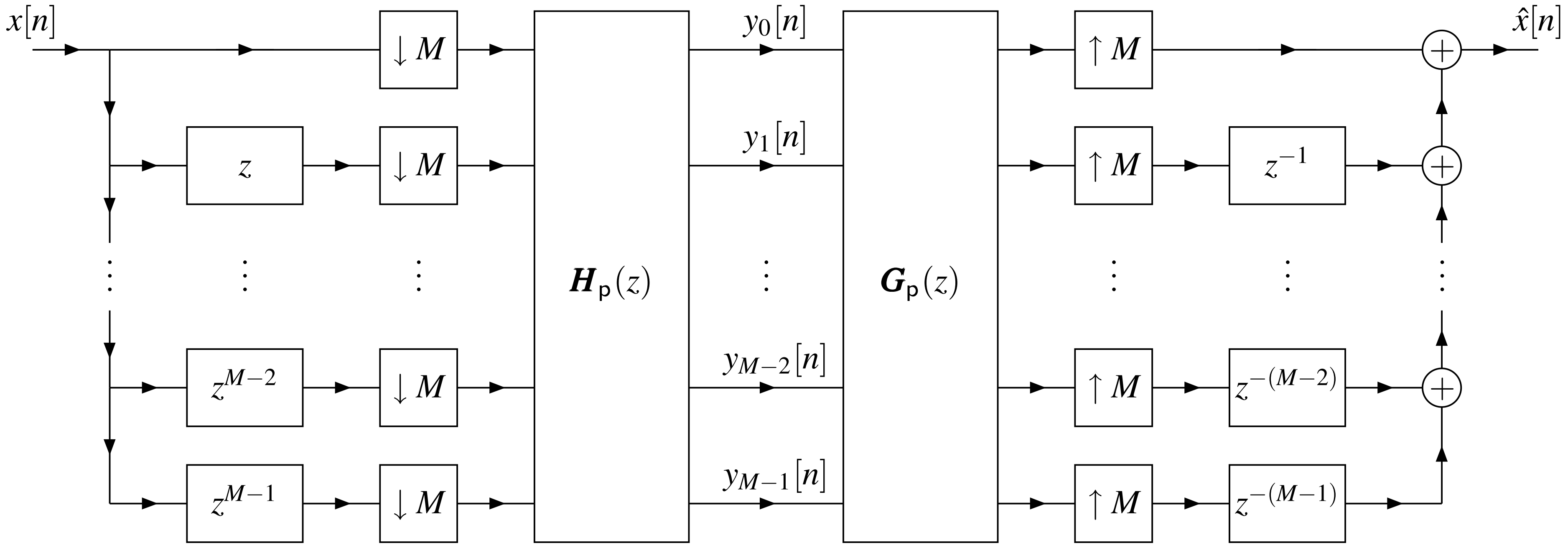
Type (1,2) Representation of UMD Filter Bank



$(1,2)$-type polyphase representation

- In practice, the delays in the polyphase decomposition are implemented by a ***tapped delay line***.
- Similarly, the polyphase recomposition is implemented by a ***delay/adder chain***.

Type (3,1) Representation of UMD Filter Bank



$(3,1)$-type polyphase representation

- In practice, the advances in the polyphase decomposition are implemented by a ***tapped advance line***.
- Similarly, the polyphase recomposition is implemented by a ***delay/adder chain***.

Pseudocirculant Matrices

- A matrix $\boldsymbol{P}(z)$ that is formed by taking a circulant matrix and multiplying each element below its main diagonal by z is said to be an **A-type pseudocirculant matrix**.
- A matrix $\boldsymbol{P}(z)$ that is formed by taking a circulant matrix and multiplying each element below its main diagonal by z^{-1} is said to be a **B-type pseudocirculant matrix**.
- For example, the following two matrices are, respectively, A- and B-type pseudocirculant:

$$\begin{bmatrix} P_0(z) & P_1(z) & P_2(z) \\ zP_2(z) & P_0(z) & P_1(z) \\ zP_1(z) & zP_2(z) & P_0(z) \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} P_0(z) & P_1(z) & P_2(z) \\ z^{-1}P_2(z) & P_0(z) & P_1(z) \\ z^{-1}P_1(z) & z^{-1}P_2(z) & P_0(z) \end{bmatrix}.$$

- Each successive row in a pseudocirculant matrix is the ***(right) circular shift*** of the preceding row with the ***circulated element multiplied by z or z^{-1}*** for an A- or B-type pseudocirculant, respectively.
- Each successive column in a pseudocirculant matrix is the ***downward circular shift*** of the preceding column with the ***circulated element multiplied by z^{-1} or z*** for an A- or B-type pseudocirculant, respectively.

Characterizing Pseudocirculant Matrices

- A pseudocirculant matrix is ***completely characterized*** by the elements of its top row.
- In particular, an $M \times M$ pseudocirculant matrix $\boldsymbol{P}(z)$ can be expressed as

$$\boldsymbol{P}(z) = \begin{cases} \displaystyle\sum_{k=0}^{M-1} P_k(z) \begin{bmatrix} \mathbf{0} & \boldsymbol{I}_{M-1} \\ z & \mathbf{0} \end{bmatrix}^k & \text{for A type} \\ \displaystyle\sum_{k=0}^{M-1} P_k(z) \begin{bmatrix} \mathbf{0} & \boldsymbol{I}_{M-1} \\ z^{-1} & \mathbf{0} \end{bmatrix}^k & \text{for B type,} \end{cases}$$

where $P_k(z)$ denotes the kth element in the top row of $\boldsymbol{P}(z)$.

Polyphase-Domain Condition for Alias-Free Systems

- **Necessary and sufficient conditions for alias cancellation.** An M-channel UMD filter bank in either $(1,2)$ or $(3,1)$ polyphase form with analysis polyphase matrix $\boldsymbol{H}_\text{p}(z)$ and synthesis polyphase matrix $\boldsymbol{G}_\text{p}(z)$ is ***alias free*** if and only if the product $\boldsymbol{P}(z) \triangleq \boldsymbol{G}_\text{p}(z)\boldsymbol{H}_\text{p}(z)$ is such that

  $$\boldsymbol{P}(z) \text{ is } \begin{cases} \text{B-type pseudocirculant} & \text{for } (1,2) \text{ type} \\ \text{A-type pseudocirculant} & \text{for } (3,1) \text{ type.} \end{cases}$$

  If the system is alias free, it has the ***transfer function***

  $$T(z) = \begin{cases} z^{-(M-1)} \sum_{k=0}^{M-1} z^{-k} P_k(z^M) & \text{for } (1,2) \text{ type} \\ \sum_{k=0}^{M-1} z^k P_k(z^M) & \text{for } (3,1) \text{ type} \end{cases}$$

  where $P_0(z), P_1(z), \ldots, P_{M-1}(z)$ are the elements of the top row of $\boldsymbol{P}(z)$.
- The top row of $\boldsymbol{P}(z)$ contains the ***polyphase components*** of $T(z)$.

Polyphase-Domain Condition for PR Systems

- **Necessary and sufficient conditions for PR.** An M-channel UMD filter bank in either $(1,2)$ or $(3,1)$ polyphase form with analysis polyphase matrix $\boldsymbol{H}_{\mathrm{p}}(z)$ and synthesis polyphase matrix $\boldsymbol{G}_{\mathrm{p}}(z)$ has the ***PR property*** if and only if the product $\boldsymbol{P}(z) \triangleq \boldsymbol{G}_{\mathrm{p}}(z)\boldsymbol{H}_{\mathrm{p}}(z)$ has the form
$$\boldsymbol{P}(z) = \begin{cases} \begin{bmatrix} \mathbf{0} & \boldsymbol{I}_{M-1} \\ z^{-1} & \mathbf{0} \end{bmatrix}^K & \text{for } (1,2) \text{ type} \\ \begin{bmatrix} \mathbf{0} & z^{-1} \\ \boldsymbol{I}_{M-1} & \mathbf{0} \end{bmatrix}^K & \text{for } (3,1) \text{ type} \end{cases}$$
for some $K \in \mathbb{Z}$. If this condition is satisfied, the relationship between the input signal x and the reconstructed signal $\hat{x}$ is given by
$$\hat{x}[n] = x[n-n_0] \quad \text{where} \quad n_0 = \begin{cases} K+M-1 & \text{for } (1,2) \text{ type} \\ K & \text{for } (3,1) \text{ type.} \end{cases}$$
Thus, the ***shift-free PR property*** holds if and only if
$$\boldsymbol{P}(z) = \begin{cases} \begin{bmatrix} \mathbf{0} & z\boldsymbol{I}_{M-1} \\ 1 & \mathbf{0} \end{bmatrix} & \text{for } (1,2) \text{ type} \\ \boldsymbol{I} & \text{for } (3,1) \text{ type.} \end{cases}$$

Comment on Design of PR Systems

- In the condition characterizing the PR property (on the previous slide), if $K = 0$,

$$\boldsymbol{P}(z) \triangleq \boldsymbol{G}_{\mathrm{p}}(z)\boldsymbol{H}_{\mathrm{p}}(z) = \boldsymbol{I}.$$

- Since the identity matrix is sometimes an attractive matrix with which to work, one might wonder what degree of freedom is lost by constraining K to be zero.
- As it turns out, not much is sacrificed, as changing K only serves to introduce additional delay into the analysis/synthesis filters.
- Consequently, we often only consider the case of $\boldsymbol{P}(z) = \boldsymbol{I}$ when designing PR filter banks.
- Delay/advance can always be added to the analysis and synthesis filters, after the fact, if required.
- With $\boldsymbol{P}(z) = \boldsymbol{I}$, the problem of designing PR filter banks, in some sense, reduces to a problem of ***factorizing the identity matrix***.
- Many design methods for PR filter banks have been developed and are often based on optimization techniques.

Polyphase-Domain Condition for PR Systems Revisited

- **Theorem.** Any M-channel PR UMD filter bank in either $(1,2)$ or $(3,1)$ polyphase form with analysis polyphase matrix $\boldsymbol{H}_{\text{p}}(z)$ and synthesis polyphase matrix $\boldsymbol{G}_{\text{p}}(z)$ must be such that the product $\boldsymbol{P}(z) \triangleq \boldsymbol{G}_{\text{p}}(z)\boldsymbol{H}_{\text{p}}(z)$ has a ***determinant of the form***

  $$\det \boldsymbol{P}(z) = (-1)^{K(M-1)} z^{-K},$$

  where K is an integer. Moreover, if the analysis and synthesis filters of the UMD filter bank are of the ***FIR type***, this condition implies that the polyphase matrices must have ***determinants of the form***

  $$\det \boldsymbol{H}_{\text{p}}(z) = \alpha_0 z^{-L_0} \quad \text{and} \quad \det \boldsymbol{G}_{\text{p}}(z) = \alpha_1 z^{-L_1},$$

  where the α_i are nonzero constants and the L_i are integers. That is, the determinants of the polyphase matrices must be ***monomials***.
- The above result regarding FIR filters is particularly useful, as it provides a means to test if, for a given set of FIR analysis filters, there exists a set of FIR synthesis filters that yield PR.

UMD Filter Banks and Linear Transformations

- An M-channel UMD filter bank can be viewed as a ***pair of linear transformations***.
- The ***analysis side*** is associated with a ***linear transformation*** $\mathcal{H}$ that maps the input x to the subband signals $\{y_k\}_{k=0}^{M-1}$.
- The ***synthesis side*** is associated with a ***linear transformation*** $\mathcal{G}$ that maps the subband signals $\{y_k\}_{k=0}^{M-1}$ to the output $\hat{x}$.
- A shift-free PR filter bank is a filter bank satisfying $\mathcal{G}\mathcal{H} = \mathcal{I}$, where $\mathcal{I}$ denotes the identity (i.e., $\mathcal{G}$ is the inverse of $\mathcal{H}$). Such a filter bank is called **biorthonormal** (or **biorthogonal**).
- If a filter bank not only satisfies $\mathcal{G}\mathcal{H} = \mathcal{I}$ but $\mathcal{G}$ and $\mathcal{H}$ are also orthogonal transformations (i.e., $\mathcal{G}^{-1} = \mathcal{G}^T$ and $\mathcal{H}^{-1} = \mathcal{H}^T$), the filter bank is called **orthonormal**.
- The orthonormal property is often desirable, as it ensures that the total energy in the subband signals $\{y_k\}_{k=0}^{M-1}$ equals the energy in the input x (i.e., ***energy is preserved***).
- In practical terms, this energy-preservation property leads to systems that are inherently more ***numerically robust***.

Filter Banks and Basis Representations

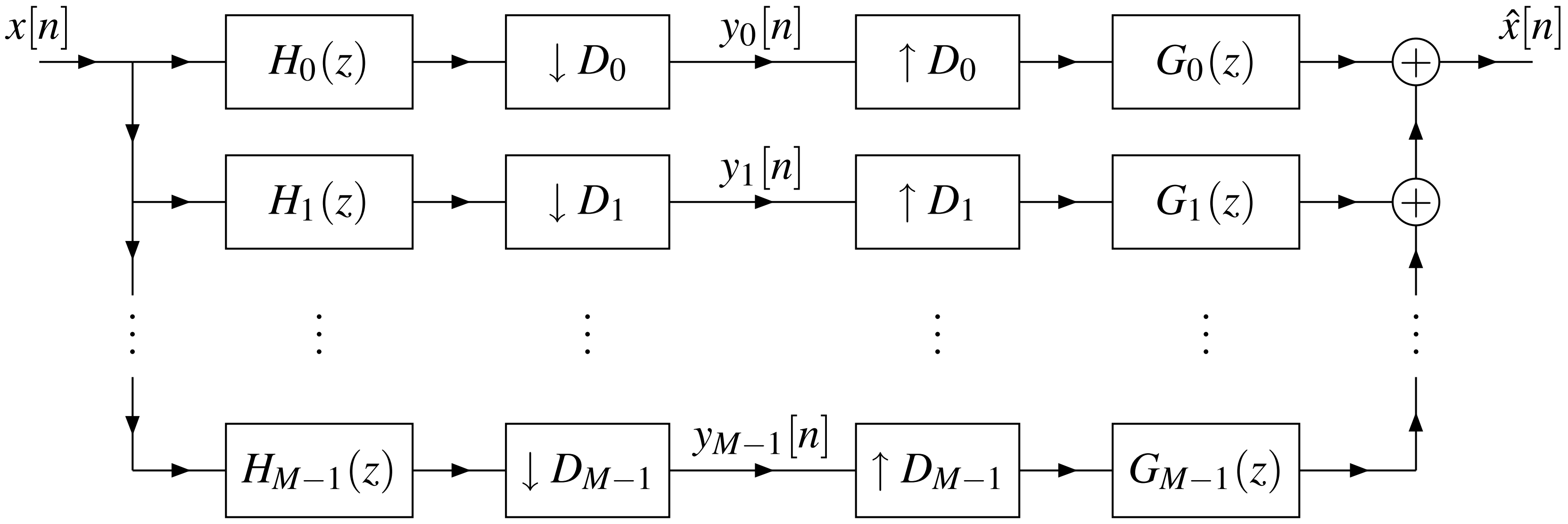
- For a sequence x, we often employ an expansion of the form
$$x[n] = \sum_k a_k \varphi_k[n],$$
where $\{a_k\}$ are expansion coefficients and $\{\varphi_k\}$ are basis vectors.
- For example, an N-periodic sequence x can be represented by a discrete-time Fourier series (DTFS), which chooses
$$\varphi_k[n] = e^{jk2\pi n/N},$$
leading to the representation
$$x[n] = \sum_{k=0}^{N-1} a_k e^{jk2\pi n/N} \quad \text{where} \quad a_k = \tfrac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-jk2\pi n/N}.$$
- Let x denote the input to a shift-free PR filter bank with subband sequences $\{y_k\}_{k=0}^{M-1}$.
- The filter bank is associated with a basis representation of x.
- The elements of the sequences $\{y_k\}_{k=0}^{M-1}$ correspond to the $\{a_k\}$.
- The synthesis side of the filter bank produces x from $\{a_k\}$, with $\{\varphi_k\}$ being determined by the synthesis-filter impulse responses.

Orthonormal UMD Filter Banks

- **Orthonormality conditions.** Suppose that we have an M-channel UMD filter bank with analysis filters $\{H_k(z)\}$, synthesis filters $\{G_k(z)\}$, analysis polyphase matrix $\boldsymbol{H}_\mathrm{p}(z)$, and synthesis polyphase matrix $\boldsymbol{G}_\mathrm{p}(z)$. Let $h_k[n]$ and $g_k[n]$ denote the inverse $\mathcal{Z}$ transforms of $H_k(z)$ and $G_k(z)$, respectively. Then, the above system is ***orthonormal*** if and only if the system has the shift-free PR property and
$$h_k[n] = g_k^*[-n].$$
Furthermore, this condition is ***equivalent*** to each of the following:
$$\boldsymbol{G}_\mathrm{m}(z)\boldsymbol{G}^T_{\mathrm{m}*}(z^{-1}) = M\boldsymbol{I}, \quad \boldsymbol{H}_\mathrm{m}(z) = \boldsymbol{G}^T_{\mathrm{m}*}(z^{-1}); \text{ and}$$
$$\boldsymbol{G}_\mathrm{p}(z)\boldsymbol{G}^T_{\mathrm{p}*}(z^{-1}) = \begin{cases} \begin{bmatrix} \boldsymbol{0} & z\boldsymbol{I}_{M-1} \\ 1 & \boldsymbol{0} \end{bmatrix} & \text{for } (1,2) \text{ type} \\ \boldsymbol{I} & \text{for } (3,1) \text{ type,} \end{cases} \qquad \boldsymbol{H}_\mathrm{p}(z) = \boldsymbol{G}^T_{\mathrm{p}*}(z^{-1}).$$
- The condition on $\boldsymbol{G}_\mathrm{p}$ in the (3,1) case is equivalent to $\boldsymbol{G}_\mathrm{p}$ being paraunitary.
- For this reason, an orthonormal filter bank is often called **paraunitary**.

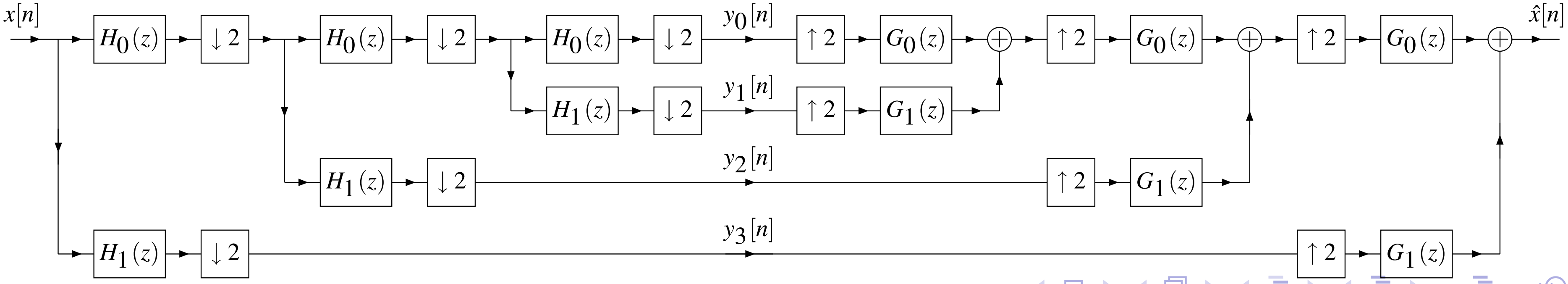
Nonuniformly-Decimated Filter Banks

- If we modify a UMD filter bank so that the various channels do not all use the same downsampling/upsampling factor, we obtain what is called a **nonuniformly-decimated** filter bank, as shown below.



Tree-Structured Filter Banks

- The analysis side of an M-channel UMD filter bank decomposes the input signal x into M subband signals $\{y_k\}_{k=0}^{M-1}$, and the synthesis side then recombines these subband signals to obtain the output $\hat{x}$.
- There is nothing, however, to prevent us from using additional UMD filter banks to further decompose some or all of the subband signals. In other words, the process of splitting a signal into subbands can be applied recursively.
- This recursive splitting process leads to a filter bank with a tree structure called a **tree-structured filter bank**.
- An example of a tree-structured filter bank obtained by recursively decomposing the lowpass subband signal of a two-channel UMD filter bank is shown below.



Tree-Structured Filter Banks and Discrete Wavelet Transforms

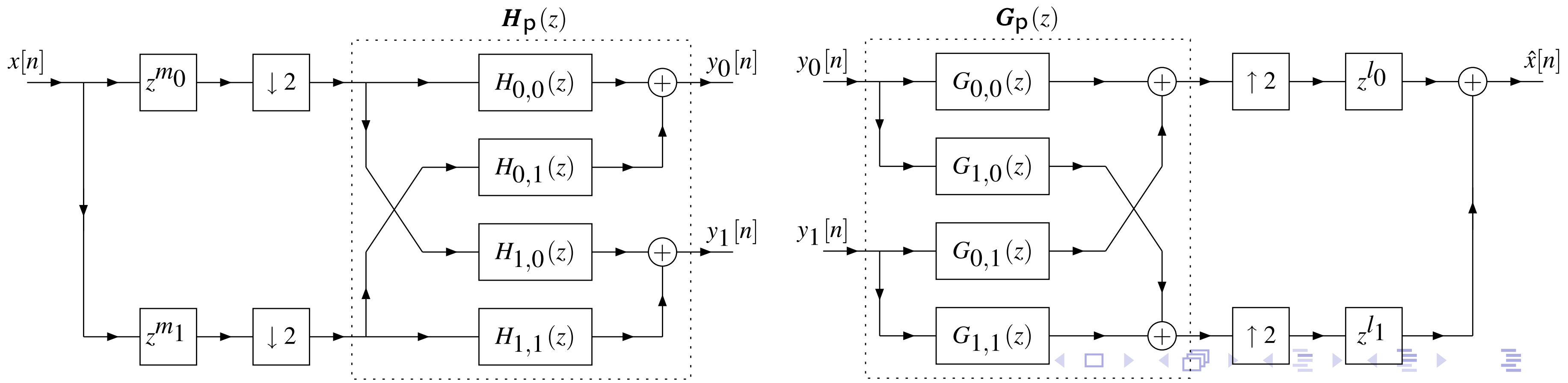
- Discrete wavelet transforms (DWTs) are computed by tree-structured filter banks.
- In particular, if a tree-structured filter bank is such that the following conditions are satisfied, then the tree-structured filter bank can be shown to compute a DWT:
  1. only the ***lowpass*** subband signal is decomposed at each level in the tree;
  2. the ***same*** basic UMD filter bank building block is used for decomposition at all levels; and
  3. the basic block has ***shift-free PR*** and satisfies certain ***regularity conditions***.
- The analysis and synthesis sides of the tree-structured filter bank computes the forward and inverse DWTs, respectively.
- Since UMD filter banks are the basic building block of DWTs, UMD filter banks play a crucial role in the context of wavelet systems.

Section 3.3

Implementation of UMD Filter Banks

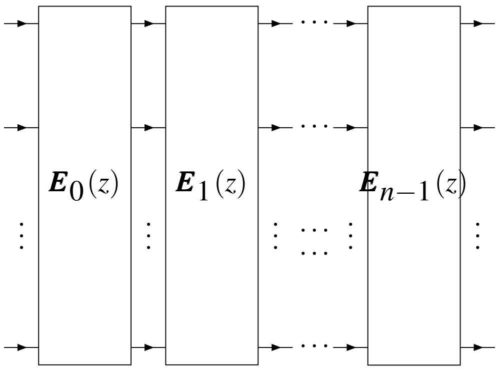
Implementation of UMD Filter Banks

- In practice, for reasons of efficiency, filter banks are always implemented in polyphase form.
- For a filter bank in polyphase form, each of the analysis and synthesis filtering is represented by an M-input M-output LTI network, which is comprised of M^2 filters.
- Although each network could be implemented directly in terms of these M^2 filters, this not usually desirable.
- Such a direct implementation would have relatively high-order filters and a very complicated irregular structure, which is difficult to implement efficiently.
- If $M = 2$, we have 4 filters with the somewhat complicated interconnection pattern shown, and the problem quickly worsens as M grows:

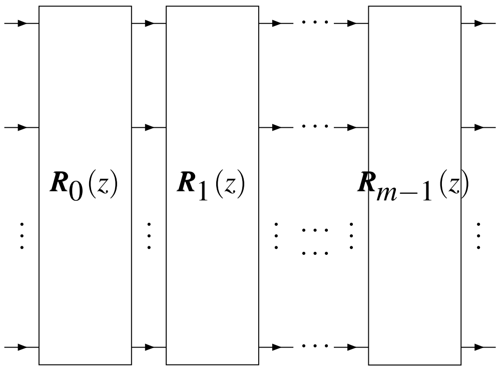


Block Cascade Realization

- To avoid unnecessarily complicated implementations, it is beneficial to break the polyphase filtering into a number of simpler cascaded stages.
- Instead of implementing $\boldsymbol{H}_{\mathrm{p}}(z)$ directly, we further decompose $\boldsymbol{H}_{\mathrm{p}}(z)$ as follows
$$\boldsymbol{H}_{\mathrm{p}}(z) = \boldsymbol{E}_{n-1}(z)\cdots\boldsymbol{E}_1(z)\boldsymbol{E}_0(z).$$
- Each of the $\{\boldsymbol{E}_i(z)\}$ can then be taken to represent a single filtering stage in the final implementation as shown below.
- Similarly, $\boldsymbol{G}_{\mathrm{p}}(z)$ can be decomposed to produce
$$\boldsymbol{G}_{\mathrm{p}}(z) = \boldsymbol{R}_{m-1}(z)\cdots\boldsymbol{R}_1(z)\boldsymbol{R}_0(z).$$
- This corresponds to the cascade realization of the synthesis polyphase matrix shown below.



Block cascade realization of analysis polyphase filtering



Block cascade realization of synthesis polyphase filtering

Block Cascade Realization (Continued)

- Different factorizations of the analysis and synthesis polyphase matrices correspond to distinct block-cascade realizations of the analysis and synthesis polyphase filtering.
- To obtain regular structures, we choose the factors such that they follow some pattern.
- For example, we might choose the $\{\boldsymbol{E}_k(z)\}_{k=0}^{n-1}$ and $\{\boldsymbol{R}_k(z)\}_{k=0}^{m-1}$ such that they are all triangular or diagonal matrices.

Type-S Elementary Matrices

- **Type-S matrix**: This type of $M \times M$ matrix has all of the elements on the main diagonal equal to one, except the (k,k) entry which is α and all off-main-diagonal elements equal to zero.
- Such a matrix is completely characterized by the pair (k,α) and is denoted $\mathcal{S}_M(k,\alpha)$. In cases where the size of the matrix is clear from the context, the subscript M is omitted.
- Note that $\mathcal{S}^{-1}(k,\alpha) = \mathcal{S}(k,\alpha^{-1})$. That is, the inverse of a type S matrix is another type S matrix.
- For example, we have:

$$\mathcal{S}_3(1,\alpha) = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathcal{S}_3^{-1}(1,\alpha) = \begin{bmatrix} \alpha^{-1} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Type-A Elementary Matrices

- **Type-A matrix**: This type of $M \times M$ matrix has all ones on the main diagonal, α at position (k,l), and all other entries zero.
- Such a matrix is completely characterized by the triple (k,l,α) and is denoted $\mathcal{A}_M(k,l,\alpha)$. The subscript M may be omitted in cases where the size of the matrix is clear from the context.
- Note that $\mathcal{A}^{-1}(k,l,\alpha) = \mathcal{A}(k,l,-\alpha)$. That is, the inverse of a type A matrix is another type A matrix.
- For example, we have:

$$\mathcal{A}_3(1,2,1+z) = \begin{bmatrix} 1 & 1+z & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathcal{A}_3^{-1}(1,2,1+z) = \begin{bmatrix} 1 & -1-z & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Lifting Realization

- In a lifting realization, a UMD filter bank is implemented in its ***polyphase form***, using a ***block cascade*** realization of the polyphase filtering networks.
- The distinguishing characteristic of this realization is the type of network used to implement the polyphase filtering.
- A ***lifting factorization*** of the polyphase matrices is used.
- The lifting factorization decomposes a matrix into zero or more constant type S factors that either premultiply or postmultiply zero or more type A factors.

Lifting Realization (Continued)

- The analysis polyphase matrix $\boldsymbol{H}_{\mathrm{p}}(z)$ is decomposed using a lifting factorization as

$$\boldsymbol{H}_{\mathrm{p}}(z) = \boldsymbol{S}_{\sigma-1} \cdots \boldsymbol{S}_1 \boldsymbol{S}_0 \boldsymbol{A}_{\lambda-1}(z) \cdots \boldsymbol{A}_1(z) \boldsymbol{A}_0(z)$$

  where the $\boldsymbol{S}_k$ are type S matrices with all constant entries and the $\boldsymbol{A}_k(z)$ are type A elementary matrices (which can depend on z).

- The decomposition of the synthesis polyphase matrix $\boldsymbol{G}_{\mathrm{p}}(z)$ is completely determined by the decomposition used for the analysis polyphase matrix $\boldsymbol{H}_{\mathrm{p}}(z)$ and is given by

$$\boldsymbol{G}_{\mathrm{p}}(z) = \boldsymbol{A}_0^{-1}(z) \boldsymbol{A}_1^{-1}(z) \cdots \boldsymbol{A}_{\lambda-1}^{-1}(z) \boldsymbol{S}_0^{-1} \boldsymbol{S}_1^{-1} \cdots \boldsymbol{S}_{\sigma-1}^{-1}$$

- Furthermore, this choice of decomposition for $\boldsymbol{G}_{\mathrm{p}}(z)$ implies that $\boldsymbol{G}_{\mathrm{p}}(z) = \boldsymbol{H}_{\mathrm{p}}^{-1}(z)$.

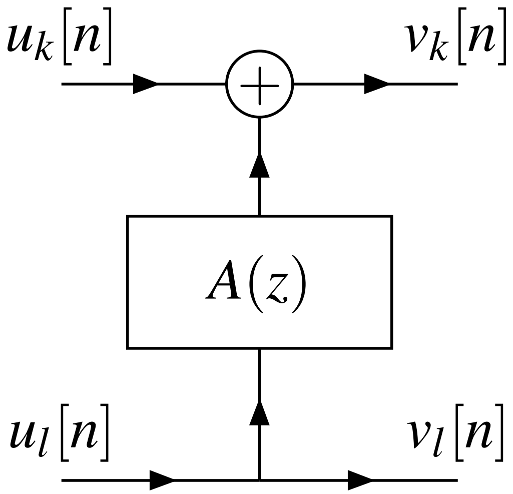
- The $\boldsymbol{A}_k^{-1}(z)$ are type A matrices and the $\boldsymbol{S}_k^{-1}$ are type S matrices. Hence, the decomposition given for $\boldsymbol{G}_{\mathrm{p}}(z)$ is, in fact, a lifting factorization of the matrix.

Existence of Lifting Realization

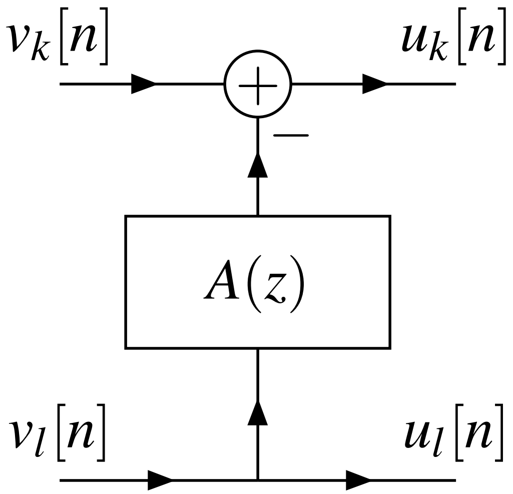
- From the definition of the lifting realization, it follows that such a realization exists if and only if a lifting factorization of $\boldsymbol{H}_{\mathrm{p}}(z)$ exists and $\boldsymbol{G}_{\mathrm{p}}(z) = \boldsymbol{H}_{\mathrm{p}}^{-1}(z)$.
- Any UMD filter bank can be made to satisfy these conditions through an appropriate normalization of its analysis and synthesis filters.

Lifting Step

- Each of the type A factors corresponds to a block that adds a filtered version of a signal in one channel to a signal in another channel.
- Such a block is called a **lifting step** and is shown below.
- To simplify the diagram only the kth and lth inputs and outputs are shown. All other inputs pass directly through to their corresponding outputs unchanged.
- The inverse of a lifting step is another lifting step.



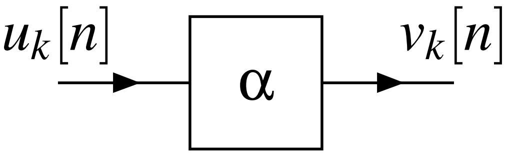
Lifting Step



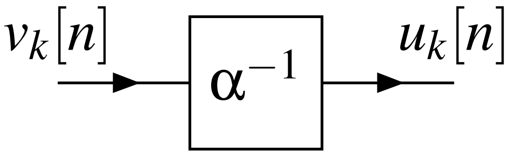
Inverse Lifting Step

Scaling Step

- Each of the type S factors corresponds to a block that scales the signal in a single channel.
- Such a block is called a **scaling step** and is shown below.
- Only the kth input and output are shown as all other inputs pass directly through to their corresponding outputs without modification.
- The inverse of a scaling step is another scaling step.



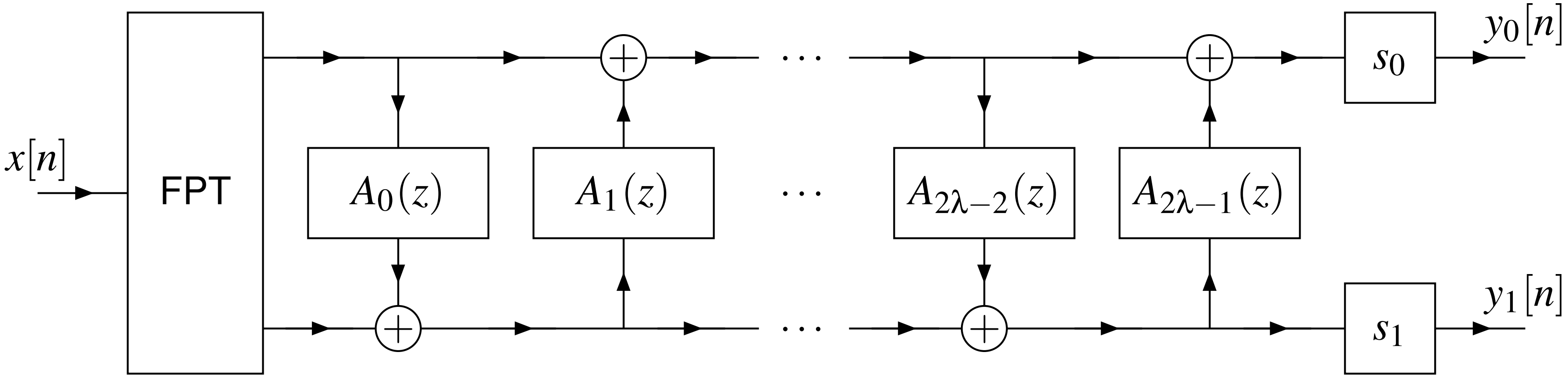
Scaling Step



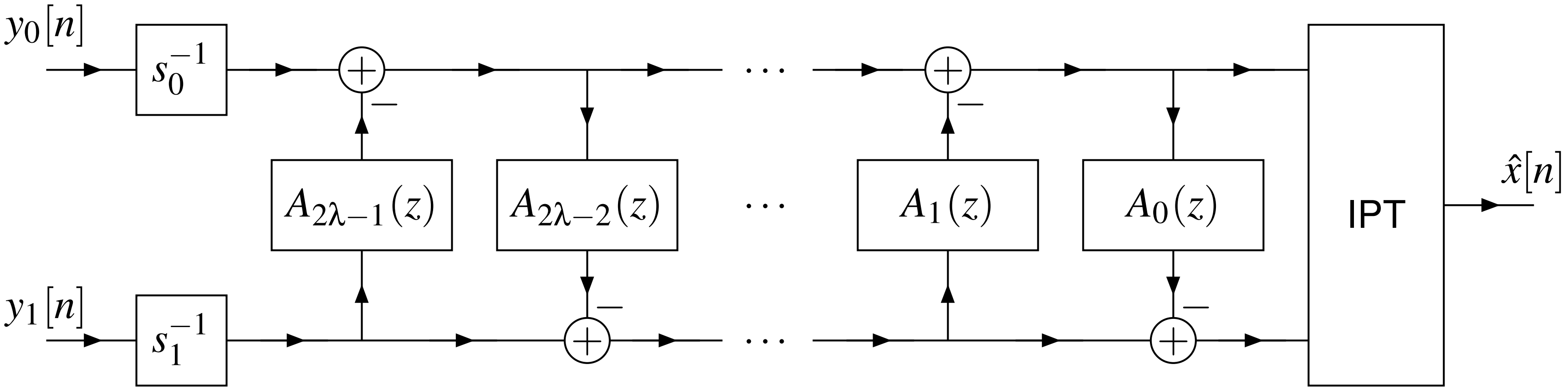
Inverse Scaling Step

Lifting Realization: Two-Channel Case

- Notice the high degree of symmetry between the analysis and synthesis sides.

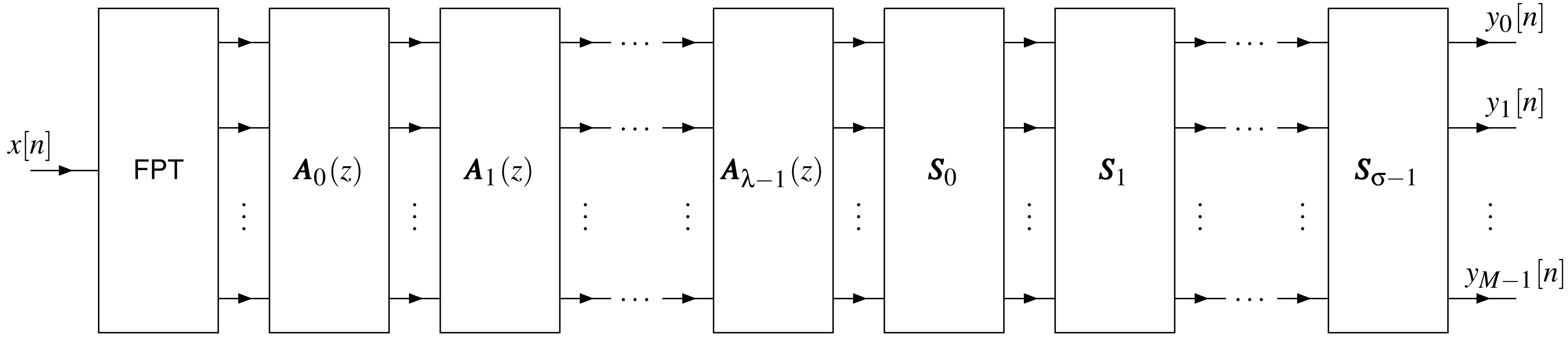


Analysis Side

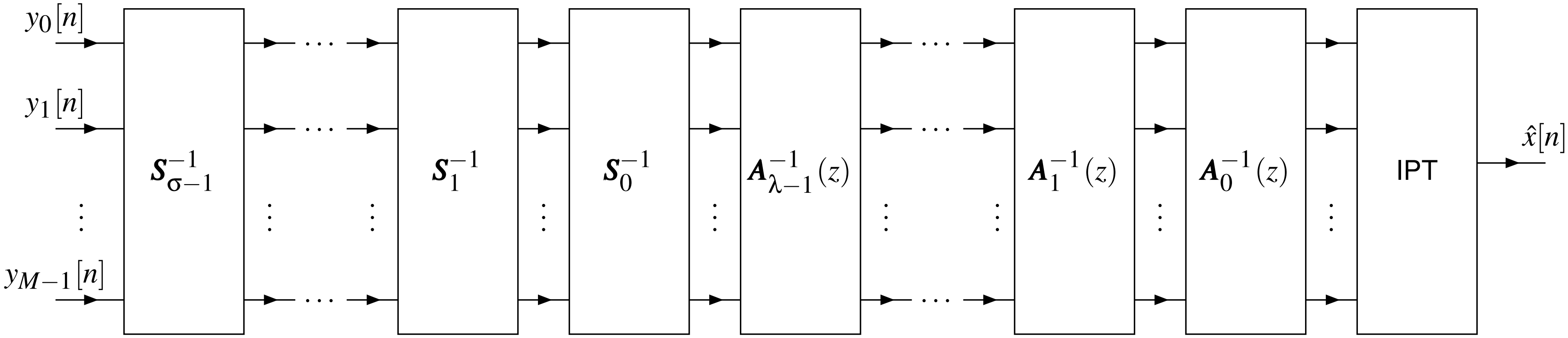


Synthesis Side

Lifting Realization: M-Channel Case



Analysis Side



$y_0[n]$
$y_1[n]$
$y_{M-1}[n]$
$\boldsymbol{S}_{\sigma-1}^{-1}$
$\boldsymbol{S}_1^{-1}$
$\boldsymbol{S}_0^{-1}$
$\boldsymbol{A}_{\lambda-1}^{-1}(z)$
$\boldsymbol{A}_1^{-1}(z)$
$\boldsymbol{A}_0^{-1}(z)$
IPT
$\hat{x}[n]$

Synthesis Side

Advantages of Lifting Realization

- The transform can be calculated ***in place*** without the need for auxiliary memory.
- Even in the presence of ***coefficient quantization error***, the PR property is retained.
- The PR property can be maintained even in the presence of the ***roundoff error*** introduced by finite precision arithmetic (if scaling steps use nonzero integer factors).
- The lifting realization can be used to easily construct ***reversible*** transforms.
- The inverse transform has the ***same computational complexity*** as the forward transform.
- Asymptotically, for long filters, the lifting realization yields a ***more computationally efficient*** structure than the standard realization.
- The PR property is ***structurally imposed***, regardless of the choice of lifting filters.

Section 3.4

UMD Filter Banks: Practical Issues

Filter Banks and Multidimensional Signals

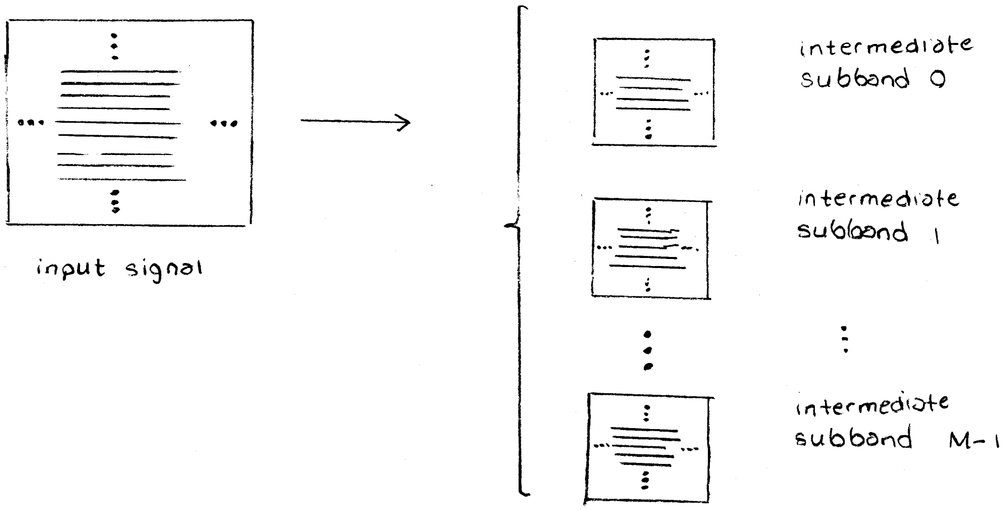
- The UMD filter banks that we have considered so far are fundamentally ***one-dimensional*** in nature.
- Many types of signals, however, are ***multidimensional*** in nature (e.g., images, video, volumetric medical data, and so on).
- As it turns out, the easiest way in which to construct a multidimensional filter bank is from ***one-dimensional building blocks***.
- In other words, we construct a multidimensional filter bank as a ***composition*** of one-dimensional filter banks.
- Or put another way, we view a multidimensional signal as being comprised of ***one-dimensional slices***, which are then processed with one-dimensional filter banks.

Processing of Two-Dimensional Signals

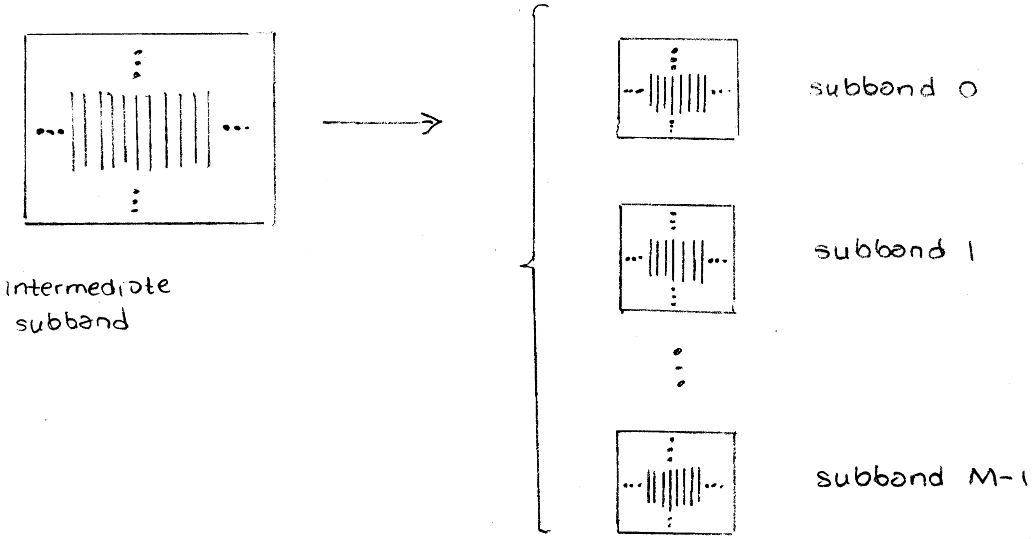
- Consider a two-dimensional signal x.
- The kth (one-dimensional) ***horizontal slice*** of x is given by $x_{\mathrm{h},k}[n] = x[n,k]$ and the kth (one-dimensional) ***vertical slice*** of x is given by $x_{\mathrm{v},k}[n] = x[k,n]$.
- To begin, we apply a (one-dimensional) filter bank to each horizontal slice of x.
- For each horizontal slice that is processed, M one-dimensional subband signals are produced.
- Then, for each channel, we vertically stack the one-dimensional subband signals to produce M two-dimensional signals.
- Next, for each of the M two-dimensional subband signals, we apply the (one-dimensional) filter bank to each vertical slice.
- This yields M new subband signals for each of the M original subband signals, for a total of M^2 subbands.

Processing Two-Dimensional Signals (Continued)

1. process horizontal slices of signal:

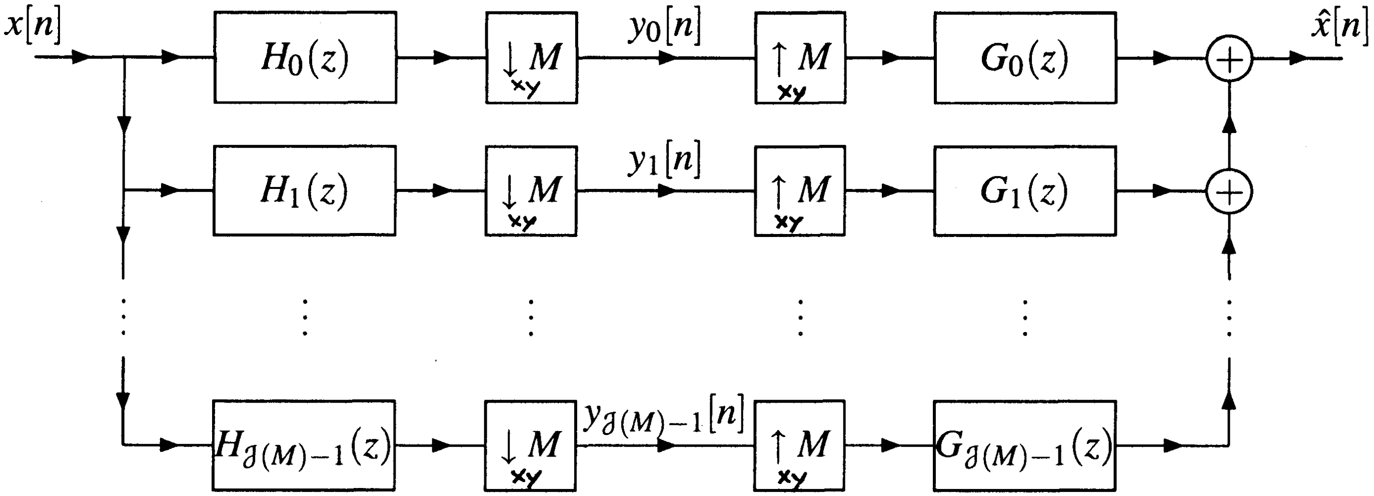


2. process vertical slices of intermediate subband signals:



Separable Two-Dimensional UMD Filter Banks

- In effect, we have constructed the M^2-channel two-dimensional filter bank, shown below, where $\mathcal{J}(M) = M^2$.

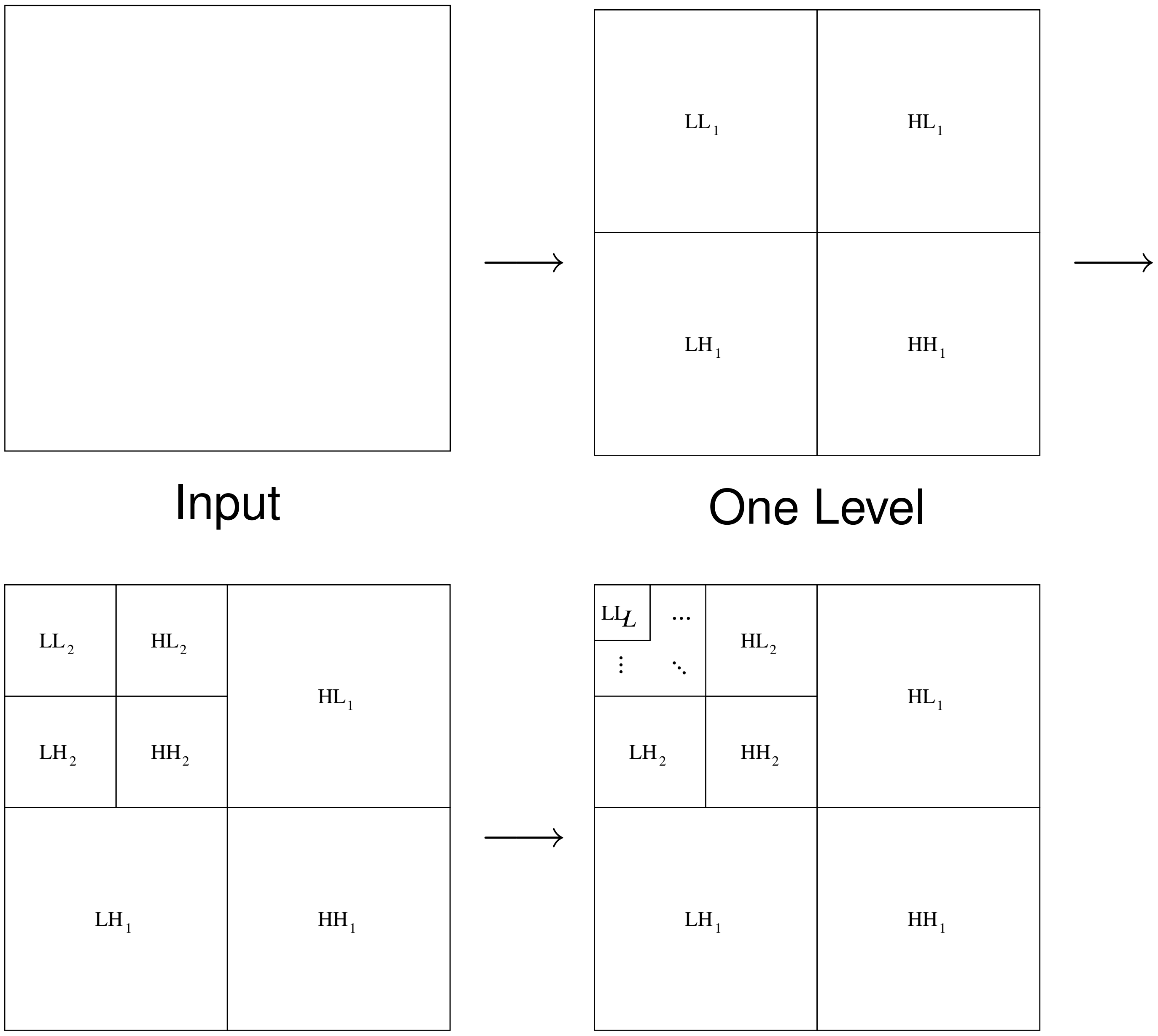


- The filters $\{H_k\}_{k=0}^{M^2-1}$ and $\{G_k\}_{k=0}^{M^2-1}$ employ two-dimensional filtering operations that are composed from one-dimensional operations, and the two-dimensional downsamplers/upsamplers are composed from one-dimensional downsamplers/upsamplers.
- Since the multidimensional operations can be decomposed into one-dimensional operations, the filter bank is called **separable**.
- Although we have only considered the two-dimensional case here for simplicity, this idea ***trivially extends*** to any arbitrary number of dimensions.

Advantages/Disadvantages of Separable Approach

- The separable approach to constructing multidimensional filter banks has a number of ***advantages***:
  1. It is ***conceptually simple*** and ***easy to analyze***, requiring only one-dimensional signal-processing theory.
  2. The approach is also ***computationally efficient***, as all operations are fundamentally one-dimensional in nature.
- Although the separable approach is adequate for many applications, it also has some significant ***disadvantages***:
  1. Since the multidimensional filter bank is composed from one-dimensional operations, it cannot exploit the ***true multidimensional nature*** of the signal being processed.
  2. The flexibility in the ***partitioning*** of the frequency-domain into subbands is quite limited.
  3. The ***number of channels*** possessed by the multidimensional filter bank is constrained to be M^D, which can often be overly restrictive. For example, the number of channels can become ***quite large***, depending on D and M.

Subband Structure for 2-D Wavelet Transforms



Input

One Level

Two Levels

L Levels

2-D Wavelet Transform Example

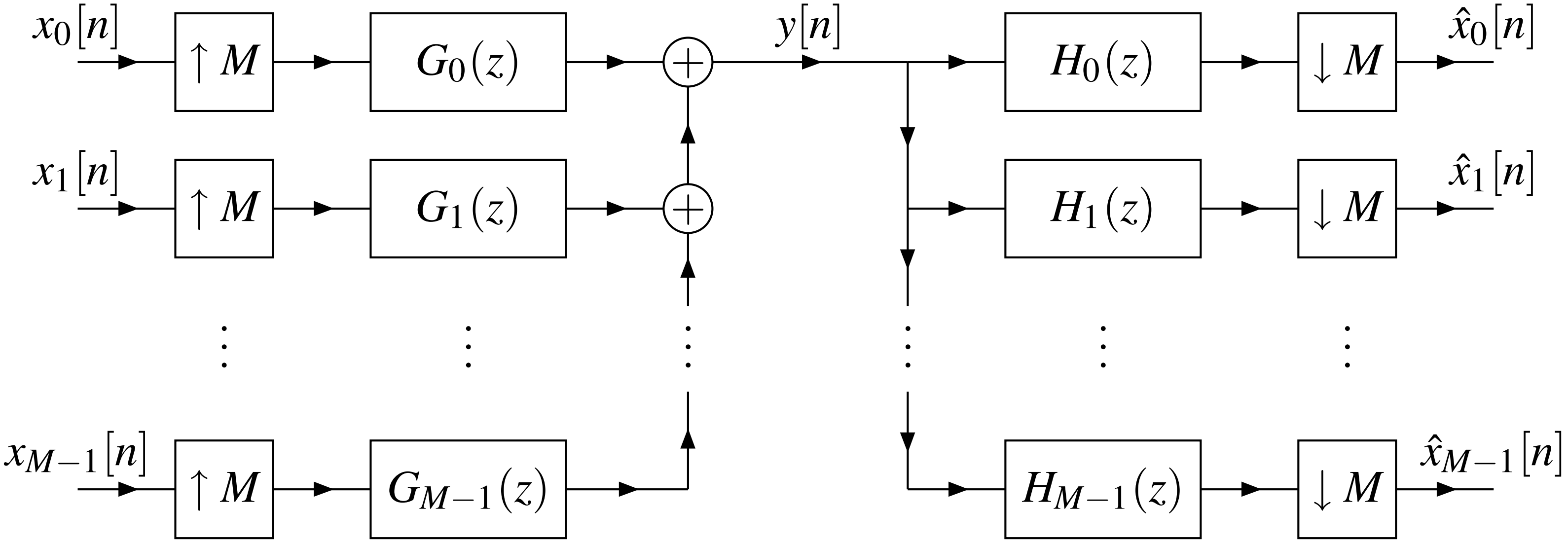
FIGURE OMITTED FOR COPYRIGHT REASONS

Section 3.5

Transmultiplexers

Transmultiplexers

- transmultiplexer is ***transpose*** of UMD filter bank (i.e., with analysis and synthesis sides swapped)
- general structure of **M-channel transmultiplexer** shown below



- transmultiplexer said to have **PR property** if, for each $k \in \{0, 1, \ldots, M-1\}$, $\hat{x}_k[n] = x_k[n-d_k]$ for some $d_k \in \mathbb{Z}$ and all $n \in \mathbb{Z}$
- transmultiplexer has PR property if corresponding UMD filter bank has PR property

Section 3.6

Applications of Multirate Systems

Sampling Rate Conversion

- sampling rate conversion by integer factor M (i.e., M-fold decimators and M-fold interpolators)
- sampling rate conversion by rational factor L/M (based on polyphase techniques)
- in many applications, need to convert between different sampling rates
- streaming video/audio at different rates
- many different sampling rates commonly used for audio/music/voice data:
  - studio recording: 44.1 kHz, 48 kHz, 88.2 kHz, 96 kHz, 192 kHz
  - MPEG-1 Audio Layer 3 (MP3): 44.1 kHz (typical), 32 kHz, 48 kHz
  - Digital Audio Tape (DAT): 48 kHz (typical), 44.1 kHz, 32 kHz
  - Compact Disc (CD): 44.1 kHz
  - DVD Audio: 44.1 kHz, 192 kHz
  - GSM-FR: 8 kHz

Part 4

Univariate Wavelet Systems

Section 4.1

Mathematical Preliminaries

Commonly-Used Sets of Functions and Sequences

- $\boldsymbol{L^2(I)}$**.** The set $L^2(I)$ is comprised of all (measurable) complex (or real) functions x defined on I for which
$$\int_I |x(t)|^2 \, dt < \infty.$$
- **Example.** The set $L^2(\mathbb{R})$ consists of all square-integrable (i.e., ***finite-energy***) complex functions defined on $\mathbb{R}$.
- $\boldsymbol{l^2(I)}$**.** The set $l^2(I)$ is comprised of all complex (or real) sequences $\{x_n\}_{n \in I}$ defined on I for which
$$\sum_{n \in I} |x_n|^2 < \infty.$$
- **Example.** The set $l^2(\mathbb{Z})$ consists of all square-summable (i.e., ***finite-energy***) complex sequences defined on $\mathbb{Z}$.

Vector Spaces

- A **vector space** over a ***scalar field*** F (such as $\mathbb{R}$ or $\mathbb{C}$) is a nonempty set V, together with two ***algebraic operations***,
  1. a mapping $(x,y) \mapsto x+y$ from $V \times V$ into V called **vector addition** and
  2. a mapping $(a,x) \mapsto ax$ from $F \times V$ into V called **scalar multiplication**,

  which satisfy the axioms of a vector space. Such a vector space is denoted $(V,F,+,\cdot)$ or simply V when the other parameters are clear from the context.
- A vector space has what is called ***algebraic structure***.
- A vector space over the field $\mathbb{R}$ is called a **real vector space**.
- A vector space over the field $\mathbb{C}$ is called a **complex vector space**.

Axioms of a Vector Space

1. for all $x, y \in V$, $x + y \in V$ (**closure under vector addition**);
2. for all $x \in V$ and all $a \in F$, $ax \in V$ (**closure under scalar multiplication**);
3. for all $x, y \in V$, $x + y = y + x$ (**commutativity of vector addition**);
4. for all $x, y, z \in V$, $(x + y) + z = x + (y + z)$ (**associativity of vector addition**);
5. for all $x \in V$ and all $a, b \in F$, $(ab)x = a(bx)$ (**associativity of scalar multiplication**);
6. for all $x \in V$ and all $a, b \in F$, $(a + b)x = ax + bx$ (**distributivity of scalar sums**);
7. for all $x, y \in V$ and all $a \in F$, $a(x + y) = ax + ay$ (**distributivity of vector sums**);
8. there exists $0 \in V$ such that $x + 0 = x$ for all $x \in V$ (**additive identity**);
9. for all $x \in V$, there exists a $(-x) \in V$ such that $x + (-x) = 0$ (**additive inverse**); and
10. for all $x \in V$, $1x = x$, where 1 denotes the multiplicative identity of the field F (**scalar multiplication identity**).

Vector Spaces

- **Example.** $\mathbb{R}^n$.
  Choose the underlying set as $V = \mathbb{R}^n$ and the field as $F = \mathbb{R}$.
  Define ***vector addition*** as:
  $(x_1, x_2, \ldots, x_n) + (y_1, y_2, \ldots, y_n) = (x_1 + y_1, x_2 + y_2, \ldots, x_n + y_n)$.
  Define ***scalar multiplication*** as: $ax = (ax_1, ax_2, \ldots, ax_n)$.
- **Example.** $l^2(\mathbb{Z})$.
  Choose the underlying set as $V = l^2(\mathbb{Z})$ and the field as $F = \mathbb{C}$.
  Define ***vector addition*** as: $(x+y)[n] = x[n] + y[n]$.
  Define ***scalar multiplication*** as: $(ax)[n] = ax[n]$.
- **Example.** $L^2(\mathbb{R})$.
  Choose the underlying set as $V = L^2(\mathbb{R})$ and the field as $F = \mathbb{C}$.
  Define ***vector addition*** as: $(x+y)(t) = x(t) + y(t)$.
  Define ***scalar multiplication*** as: $(ax)(t) = ax(t)$.

Vector Subspaces

- A subset of a vector space V that is itself a vector space is called a **vector subspace** of V.
  Example. The xy-plane is a vector subspace of $\mathbb{R}^3$.
- A subspace S of the vector space V is said to be **proper** if $S \neq V$ and **improper** if $S = V$.
- Two vector subspaces V and W of the same dimensionality are said to be **disjoint** if $V \cap W = \{0\}$ (i.e., the only common vector between V and W is the zero vector).
  Example. Let W_e and W_o denote the subspaces of $L^2(\mathbb{R})$ consisting of all even and all odd functions, respectively. Then, W_e and W_o are disjoint (since only the zero function is both even and odd).

Linear Transformations

- A transformation T of a vector space V into a vector space W, where V and W have the same scalar field F, is said to be a **linear transformation** if
  1. for all $x, y \in V$, $T(x+y) = T(x) + T(y)$ **(additivity)**; and
  2. for all $x \in V$ and all $a \in F$, $T(ax) = aT(x)$ **(homogeneity)**.
- **Example.** Some linear transformations include: scaling, rotation, shear, and reflection in $\mathbb{R}^n$, and the Fourier transform in $L^2(\mathbb{R})$ and $l^2(\mathbb{Z})$.
- The **null space** of a linear transformation $T : V \to W$, denoted $N(T)$, is the subset of V given by $N(T) = \{x \in V : Tx = 0\}$ (i.e., the set of all vectors mapped to the zero vector under the transformation T).
- The **range space** of a linear transformation $T : V \to W$, denoted $R(T)$, is defined as $R(T) = \{y = Tx : x \in V\}$ (i.e., the set of vectors produced by applying T to each of the elements of V).
- A linear transformation P of a vector space V into itself is said to be a **projection** if $P^2 = P$ (i.e., P is **idempotent**).
  Example. In $\mathbb{R}^2$, the transformation that maps (x_1, x_2) to $(x_1, 0)$ is a projection (i.e., a projection onto the x axis).

Inner Sums and Algebraic Complements

- If V and W are subspaces of the vector space U, then the **inner sum** of V and W, denoted $V+W$, is the space consisting of all points $x = v+w$ where $v \in V$ and $w \in W$. ***[Note: $V+W$ is not the same as $\neq V \cup W$.]***
  Example. Let U and V be subspaces of $\mathbb{R}^2$, where $U = \operatorname{span}\{(1,0)\}$ and $V = \operatorname{span}\{(0,1)\}$. Then, $U+V = \mathbb{R}^2$.
- Let V and W be subspaces of the vector space U. If $U = V+W$ and V and W are disjoint, W is the called the **algebraic complement** of V in U. (Similarly, V is the algebraic complement of W in U.)
  Example. Let W_e and W_o be the subspaces of $V = L^2(\mathbb{R})$ consisting of even and odd functions, respectively. Since any function can be expressed as the sum of an even and odd function, we have $W_e + W_o = L^2(\mathbb{R})$. Since W_e and W_o are disjoint, W_e and W_o are algebraic complements.
- **Theorem.** The algebraic complement always exists.
- **Theorem.** Let V and W be subspaces of a vector space U. Then for each $x \in V+W$, there is a unique $v \in V$ and a unique $w \in W$ such that $x = v+w$ if and only if V and W are disjoint (i.e., a vector has a unique decomposition in terms of algebraic complements).

Direct Sums

- Let V and W be subspaces of the vector space U. If U and V are disjoint, the (isomorphic form of the) **direct sum** of U and V, denoted $U \oplus W$, is $U + W$.

  Example. Let W_e and W_o be the subspaces of $L^2(\mathbb{R})$ consisting of all even and all odd functions, respectively. Since any function can be expressed as the sum of an even and odd function, we have $W_e + W_o = L^2(\mathbb{R})$. Since W_e and W_o are disjoint, we may also write $W_e \oplus W_o = L^2(\mathbb{R})$.

Metrics

- A **metric** d on a set X is a real function defined on $X \times X$ that satisfies the following conditions:
  1. $d(x,y) \geq 0$ for all $x,y \in X$ (**nonnegativity**);
  2. $d(x,y) = 0$ if and only if $x = y$ (**strict positivity**);
  3. $d(x,y) = d(y,x)$ for all $x,y \in X$ (**symmetry**); and
  4. $d(x,y) \leq d(x,z) + d(z,y)$ for all $x,y,z \in X$ (**triangle inequality**).
- A metric is a measure of ***distance***.
- **Example.**

  For $\mathbb{R}^2$, $d((x_1,x_2),(y_1,y_2)) = \left((x_1-y_1)^2 + (x_2-y_2)^2\right)^{1/2}$ is a metric.

  For $l^2(\mathbb{Z})$, $d(x,y) = \left(\sum_{n\in\mathbb{Z}} |x[n]-y[n]|^2\right)^{1/2}$ is a metric.

  For $L^2(\mathbb{R})$, $d(x,y) = \left(\int_{\mathbb{R}} |f(t)-g(t)|^2\, dt\right)^{1/2}$ is a metric.

Norms

- A norm on a vector space V over the field F is a mapping $\|\cdot\|$ of V into $\mathbb{R}$ with the following properties:
  1. for all $x \in V$, $\|x\| \geq 0$ (**nonnegativity**);
  2. $\|x\| = 0$ if and only if $x = 0$ (**strict positivity**);
  3. for all $x \in V$ and all $a \in F$, $\|ax\| = |a| \, \|x\|$ (**homogeneity**); and
  4. for all $x, y \in V$, $\|x + y\| \leq \|x\| + \|y\|$ (**triangle inequality**).
- A norm is a measure of ***length***.
- **Example.**

  For $\mathbb{R}^2$, $\|(x_1, x_2)\| = \left(x_1^2 + x_2^2\right)^{1/2}$ is a norm.

  For $l^2(\mathbb{Z})$, $\|x\| = \left(\sum_{n \in \mathbb{Z}} |x[n]|^2\right)^{1/2}$ is a norm.

  For $L^2(\mathbb{R})$, $\|x\| = \left(\int_{\mathbb{R}} |x(t)|^2 \, dt\right)^{1/2}$ is a norm.
- A norm induces a metric.

  Given a norm $\|\cdot\|$, the function $d(x, y) = \|x - y\|$ is a metric.

  Example. For $\mathbb{R}^2$, the function $\|(x_1, x_2)\| = \left(x_1^2 + x_2^2\right)^{1/2}$ is a norm, which induces the metric

  $d((x_1, x_2), (y_1, y_2)) = \|(x_1, x_2) - (y_1, y_2)\| = \left((x_1 - y_1)^2 + (x_2 - y_2)^2\right)^{1/2}$.

Inner Products

- An **inner product** on a vector space V over a field F is a mapping $\langle\cdot,\cdot\rangle$ of $V \times V$ into F with the following properties:
  1. $\langle x,x\rangle \geq 0$ for all $x \in V$ (**nonnegativity**);
  2. for all $x \in V$, $\langle x,x\rangle = 0$ if and only if $x = 0$ (**strict positivity**);
  3. $\langle x,y\rangle^* = \langle y,x\rangle$ for all $x,y \in V$ (**conjugate symmetry**);
  4. $\langle ax,y\rangle = a\langle x,y\rangle$ for all $x,y \in V$ and all $a \in F$ (**homogeneity**); and
  5. $\langle x+y,z\rangle = \langle x,z\rangle + \langle y,z\rangle$ for all $x,y,z \in V$ (**additivity**).
- An inner product $\langle\cdot,\cdot\rangle$ induces a norm. Given an inner product $\langle\cdot,\cdot\rangle$, the function $\|x\| = \langle x,x\rangle^{1/2}$ is a norm.
- The **angle** $\theta_{x,y}$ between two (nonzero) vectors x and y is defined as $\cos\theta_{x,y} = \frac{\langle x,y\rangle}{\|x\|\|y\|}$. [Note: $-1 \leq \frac{\langle x,y\rangle}{\|x\|\|y\|} \leq 1$.]
- An inner product facilitates the measure of angles. It imposes ***geometric structure*** on a set.
- **Example.**
  For $\mathbb{R}^n$, an inner product is $\langle(x_1,x_2,\ldots,x_n),(y_1,y_2,\ldots,y_n)\rangle = \sum_{k=1}^{n} x_k y_k$.
  For $l^2(\mathbb{Z})$, an inner product is $\langle x,y\rangle = \sum_{n\in\mathbb{Z}} x[n]y^*[n]$.
  For $L^2(\mathbb{R})$, an inner product is $\langle x,y\rangle = \int_{\mathbb{R}} x(t)y^*(t)dt$.

Inner Product Spaces and Hilbert Spaces

- A vector space V with an inner product defined on V is called an **inner product space**, and is denoted $(V, \langle\cdot,\cdot\rangle)$ or simply V when the inner product is implied from the context.
- An inner product space has ***geometric structure*** in addition to ***algebraic and topological structure***.
- A **Hilbert space** is an inner product space that satisfies a technical condition known as completeness.
- The inner product spaces used in engineering are essentially always Hilbert spaces.
- Since the inner product induces a norm, which in turn induces a metric, an inner product space has not only an inner product, but also a ***norm*** and ***metric***.

Hilbert Space $\mathbb{R}^2$

- two-dimensional Euclidean space (from high-school geometry)
- set:

  ordered pairs of real numbers (i.e., (v_1, v_2) where $v_1, v_2 \in \mathbb{R}$)
- vector addition:

  $$(v_1, v_2) + (w_1, w_2) = (v_1 + w_1, v_2 + w_2)$$
- scalar multiplication:

  $$a(v_1, v_2) = (av_1, av_2)$$
- scalar field: $\mathbb{R}$
- inner product:

  $\langle (v_1, v_2), (w_1, w_2) \rangle = (v_1, v_2) \cdot (w_1, w_2) = v_1 w_1 + v_2 w_2$ (i.e., dot product)
- norm:

  $\|(v_1, v_2)\| = \langle (v_1, v_2), (v_1, v_2) \rangle = \left(v_1^2 + v_2^2\right)^{1/2}$ (i.e., Euclidean norm)
- metric:

  $d((v_1, v_2), (w_1, w_2)) = \|(v_1, v_2) - (w_1, w_2)\| = \left((v_1 - w_1)^2 + (v_2 - w_2)^2\right)^{1/2}$
  (i.e., Euclidean distance)

Hilbert Space $l^2(\mathbb{Z})$

- finite-energy (i.e., square summable) sequences defined on $\mathbb{Z}$
- This space is normally what is used in digital signal processing.
- set:

  sequences f such that $\displaystyle\sum_{n\in\mathbb{Z}} |f[n]|^2 < \infty$
- vector addition

  $$(f+g)[n] = f[n]+g[n]$$
- scalar multiplication

  $$(af)[n] = af[n]$$
- inner product:

  $$\langle f,g\rangle = \sum_{n\in\mathbb{Z}} f[n]g^*[n]$$
- norm:

  $$\|f\| = (\langle f,f\rangle)^{1/2} = \left(\sum_{n\in\mathbb{Z}} |f[n]|^2\right)^{1/2}$$
- metric:

  $$d(f,g) = \|f-g\| = \left(\sum_{n\in\mathbb{Z}} (f[n]-g[n])^2\right)^{1/2}$$

Hilbert Space $L^2(\mathbb{R})$

- finite-energy (i.e., square integrable) functions defined on $\mathbb{R}$
- This space is normally what is used in analog signal processing.
- set:

  functions f such that $\int_{\mathbb{R}} |f(t)|^2 dt < \infty$

- vector addition:

  (f + g)(t) = f(t) + g(t)

- scalar multiplication:

  (a f)(t) = a f(t)

- inner product:

$$\langle f, g \rangle = \int_{\mathbb{R}} f(t) g^*(t) dt$$

- norm:

$$\|f\| = (\langle f, f \rangle)^{1/2} = \left(\int_{\mathbb{R}} |f(t)|^2 dt \right)^{1/2}$$

- metric:

$$d(f,g) = \|f - g\| = \left(\int_{\mathbb{R}} |f(t) - g(t)|^2 dt \right)^{1/2}$$

Orthogonality

- **Orthogonal vectors.** Two vectors x and y in an inner product space V are said to be **orthogonal**, denoted $x \perp y$, if $\langle x, y \rangle = 0$.
  Example. In $\mathbb{R}^2$, $v_1 = (1,0)$ and $v_2 = (0,1)$ are orthogonal, since $\langle v_1, v_2 \rangle = \langle (1,0),(0,1) \rangle = (1)(0) + (0)(1) = 0$.
- **Vector Orthogonal to Set.** For a subset A of an inner product space V and a vector $x \in V$, if $x \perp y$ for all $y \in A$, we say that x is orthogonal to A, denoted $x \perp A$.
  Example. In $\mathbb{R}^3$, $(0,0,1) \perp \{(1,0,0),(0,1,0),(1,1,0)\}$.
- **Set Orthogonal to Set.** For two subsets A and B of an inner product space V, if $x \perp y$ for all $x \in A$ and all $y \in B$, we say that A is orthogonal to B, denoted $A \perp B$.
  Example. In $\mathbb{R}^3$, $\{(1,0,0),(0,1,0)\} \perp \{(0,0,1),(0,0,2)\}$.
- **Orthogonal subspaces.** Two subspaces U and W of an inner product space are said to be orthogonal if $U \perp W$.

Orthogonality (Continued)

- **Orthogonal and orthonormal sequences.** A sequence of vectors $\{x_n\}_{n\in I}$ in an inner product space is said to be **orthogonal** if

$$x_n \perp x_m \text{ for all } m,n \in I,\ m \neq n,$$

and **orthonormal** if in addition to the preceding condition

$$\langle x_n, x_n\rangle = 1 \text{ for all } n \in I \text{ (i.e., } \|x_n\| = 1).$$

- **Example.** Consider the sequence $\{v_n\}_{n=1}^{2}$ of vectors in $\mathbb{R}^2$, where $v_1 = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$ and $v_2 = \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$. We have:
$\langle v_1, v_2\rangle = \left\langle \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right), \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)\right\rangle = \left(\frac{1}{\sqrt{2}}\right)\left(-\frac{1}{\sqrt{2}}\right) + \left(\frac{1}{\sqrt{2}}\right)\left(\frac{1}{\sqrt{2}}\right) = 0,$
$\langle v_1, v_1\rangle = \left\langle \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right), \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)\right\rangle = \left(\frac{1}{\sqrt{2}}\right)\left(\frac{1}{\sqrt{2}}\right) + \left(\frac{1}{\sqrt{2}}\right)\left(\frac{1}{\sqrt{2}}\right) = 1$, and
$\langle v_2, v_2\rangle = \left\langle \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right), \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)\right\rangle = \left(-\frac{1}{\sqrt{2}}\right)\left(-\frac{1}{\sqrt{2}}\right) + \left(\frac{1}{\sqrt{2}}\right)\left(\frac{1}{\sqrt{2}}\right) = 1$
Thus, $\{v_n\}_{n=1}^{2}$ is both orthogonal and orthonormal.

Orthogonal Projections

- A projection P on an inner product space is said to be **orthogonal** if its range and null spaces are orthogonal (i.e., $R(P) \perp N(P)$).
- A projection that is not orthogonal is called **oblique**.
- **Projection theorem.** If W is a closed subspace of a Hilbert space V, then every element $x \in V$ has a unique decomposition of the form $x = y + z$ where $y \in W$ and $z \in W^{\perp}$.
- **Best approximation.** Let W be a closed subspace of a Hilbert space V, and let $x \in V$. Further, let P be the orthogonal projection of V onto W. There exists a ***unique*** vector $y \in W$ that is ***closest*** to x as given by $y = Px$ (closest in the sense of minimizing $\|y - x\|$).

Orthonormal Bases

- An orthonormal sequence $\{\varphi_n)_{n\in I}$ of vectors in an inner product space V is said to be an **orthonormal basis** of V if, for every $x \in V$, there exists a ***unique*** scalar sequence $\{a_n\}_{n\in I}$ in $l^2(I)$ such that

$$x = \sum_{n\in I} a_n \varphi_n.$$

- **Example.**
  An orthonormal basis of $\mathbb{R}^3$ is given by $((1,0,0),(0,1,0),(0,0,1))$.
  An orthonormal basis of $l^2(\mathbb{Z})$ is given by $(\delta[\cdot - k])_{k\in\mathbb{Z}}$.
- **Existence of orthonormal basis.** Every Hilbert space has an orthonormal basis.

Orthonormal Bases (Continued)

- **Expansion coefficients.** Let $\{\varphi_n\}_{n \in I}$ be an orthonormal basis of an inner product space V. Then, each $x \in V$ can be expressed as $x = \sum_{n \in I} a_n \varphi_n$ where

$$a_n = \langle x, \varphi_n \rangle .$$

- **Example.** Consider the orthonormal basis (φ_1, φ_2) for $\mathbb{R}^2$, where $\varphi_1 = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$ and $\varphi_2 = (-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$. The vector $x = (2,1)$ can be expressed as

$$x = a_1 \varphi_1 + a_2 \varphi_2,$$

where

$$a_1 = \langle x, \varphi_1 \rangle = \left\langle (2,1), (\tfrac{1}{\sqrt{2}}, \tfrac{1}{\sqrt{2}}) \right\rangle = (2)(\tfrac{1}{\sqrt{2}}) + (1)(\tfrac{1}{\sqrt{2}}) = \frac{3}{\sqrt{2}} \quad \text{and}$$

$$a_2 = \langle x, \varphi_2 \rangle = \left\langle (2,1), -\tfrac{1}{\sqrt{2}}, \tfrac{1}{\sqrt{2}}) \right\rangle = (2)(-\tfrac{1}{\sqrt{2}}) + (1)(\tfrac{1}{\sqrt{2}}) = -\frac{1}{\sqrt{2}}.$$

Parseval Identity

- **Parseval identity**. Let $\{\varphi_n\}_{n \in I}$ be an orthonormal basis of a Hilbert space V. Then, for all $x, y \in V$,

  $$\langle x, y\rangle = \sum_{n \in I} \langle x, \varphi_n\rangle \langle y, \varphi_n\rangle^*,$$

  which, for $x = y$, simplifies to

  $$\|x\|^2 = \sum_{n \in I} |\langle x, \varphi_n\rangle|^2$$

  (i.e., an orthonormal basis preserves inner products and norms).
  [Note that $x = \sum_{n \in I} \langle x, \varphi_n\rangle \varphi_n$ and $y = \sum_{n \in I} \langle y, \varphi_n\rangle \varphi_n$.]
- **Example.** Let V be the subspace of $L^2(\mathbb{R})$ with the orthonormal basis $\{\varphi_n\}_{n \in \mathbb{Z}}$ where $\varphi_n(t) = \operatorname{sinc}(t - \pi n)$ and let $x \in V$ be given by $x = \sum_{n \in \mathbb{Z}} a[n]\varphi_n = 3\varphi_0 + 4\varphi_1$. Determine $\|x\|$.
  We have that $\|x\| = \|a\| = \left(\sum_{n \in \mathbb{Z}} |a[n]|^2\right)^{1/2} = (3^2 + 4^2)^{1/2}$ = 5.

Orthogonal Projections (Continued)

- **Finding orthogonal projection.** Let W be a closed subspace of an Hilbert space V, and let $(e_n)_{n \in I}$ be an orthonormal basis for W. Further, let P denote the orthogonal projection of V onto W. Then, P is given by $Px = \sum_{n \in I} \langle x, e_n \rangle e_n$, where $x \in V$.
- **Example.** Let (e_1, e_2) be an orthonormal basis for a subspace W of $\mathbb{R}^3$, where $e_1 = (\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}})$ and $e_2 = (-\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}})$. Find the orthogonal projection y of $x = (1,2,1)$ onto W. We have $y = \langle x, e_1 \rangle e_1 + \langle x, e_2 \rangle e_2 = \left\langle (1,2,1), (\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}) \right\rangle (\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}) + \left\langle (1,2,1), (-\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}) \right\rangle (-\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}) = \sqrt{2}(\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}) + 0 = (1,0,1)$.

Biorthogonality

- Two sequences $\{x_n\}_{n\in I}$ and $\{y_n\}_{n\in I}$ of vectors in an inner product space are said to be **biorthogonal** if $x_m \perp y_n$ for all $m, n \in I$, $m \neq n$. If, in addition, $\langle x_n, y_n\rangle = 1$ for all $n \in I$, then the sequences are said to be **biorthonormal**.
- An orthogonal sequence is biorthogonal with itself.
  An orthonormal sequence is biorthonormal with itself.
- **Example.** Let (φ_1, φ_2) and $(\tilde{\varphi}_1, \tilde{\varphi}_2)$ be sequences of vectors in $\mathbb{R}^2$, where $\varphi_1 = (1,1)$, $\varphi_2 = (-1,1)$, $\tilde{\varphi}_1 = (\frac{1}{2}, \frac{1}{2})$, and $\tilde{\varphi}_2 = (-\frac{1}{2}, \frac{1}{2})$. We have:

$$\begin{aligned}
\langle \varphi_1, \tilde{\varphi}_1\rangle &= (1)(\tfrac{1}{2}) + (1)(\tfrac{1}{2}) = 1, \\
\langle \varphi_2, \tilde{\varphi}_2\rangle &= (-1)(-\tfrac{1}{2}) + (1)(\tfrac{1}{2}) = 1, \\
\langle \varphi_1, \tilde{\varphi}_2\rangle &= (1)(-\tfrac{1}{2}) + (1)(\tfrac{1}{2}) = 0, \quad \text{and} \\
\langle \varphi_2, \tilde{\varphi}_1\rangle &= (-1)(\tfrac{1}{2}) + (1)(\tfrac{1}{2}) = 0.
\end{aligned}$$

  Thus, (φ_1, φ_2) and $(\tilde{\varphi}_1, \tilde{\varphi}_2)$ are both biorthogonal and biorthonormal.

Riesz Bases

- Although an orthonormal basis is often convenient to use, orthonormality can place too many constraints on the choice of basis vectors.
- If we drop the orthonormality constraint, we need to impose some condition to ensure that the basis is well behaved.
- A sequence $\{\varphi_n\}_{n\in I}$ of vectors in a Hilbert space V is said to be a **Riesz basis** of V if, for every $x \in V$, there exists a ***unique*** scalar sequence $\{a_n\}_{n\in I}$ in $l^2(I)$ such that

$$x = \sum_{n\in I} a_n \varphi_n,$$

and there exist real numbers $A, B > 0$ (independent of x) satisfying the **Riesz condition**

$$A\|a\|^2 \leq \|x\|^2 \leq B\|a\|^2$$

(or equivalently, $\frac{1}{B}\|x\|^2 \leq \|a\|^2 \leq \frac{1}{A}\|x\|^2$). The constants A and B are referred to as the lower and upper **Riesz bounds**, respectively.

- An orthonormal basis is a special case of a Riesz basis with $A = B = 1$.

Dual Riesz Bases

- Let $\{\varphi_n\}_{n\in I}$ be a Riesz basis of a Hilbert space V with lower and upper Riesz bounds A and B, respectively. Then, there exists another Riesz basis $\{\tilde{\varphi}_n\}_{n\in I}$ of V with lower and upper Riesz bounds $\frac{1}{B}$ and $\frac{1}{A}$, respectively, such that for all $x \in V$,

$$x = \sum_{n\in I} \langle x, \tilde{\varphi}_n \rangle \varphi_n = \sum_{n\in I} \langle x, \varphi_n \rangle \tilde{\varphi}_n.$$

  We call $\{\tilde{\varphi}_n\}_{n\in I}$ the **dual Riesz basis** of $\{\varphi_n\}_{n\in I}$.
- **Theorem.** Dual Riesz bases are biorthonormal.
- To compute the ***expansion coefficients*** of a vector in terms of a Riesz basis, the dual basis is used as shown above.
- **Example.** Let $\{\varphi_n\}_{n=1}^{2}$ and $\{\tilde{\varphi}_n\}_{n=1}^{2}$ be dual Riesz bases of $\mathbb{R}^2$, where $\varphi_1 = (1,1)$, $\varphi_2 = (-1,1)$, $\tilde{\varphi}_1 = (\frac{1}{2},\frac{1}{2})$, and $\tilde{\varphi}_2 = (-\frac{1}{2},\frac{1}{2})$. Express $x = (3,1)$ in terms of the basis $\{\varphi_n\}_{n=1}^{2}$. We have: $x = a_1\varphi_1 + a_2\varphi_2$, where

$$a_1 = \langle x, \tilde{\varphi}_1 \rangle = \left\langle (3,1), (\tfrac{1}{2},\tfrac{1}{2}) \right\rangle = \tfrac{3}{2} + \tfrac{1}{2} = 2 \quad \text{and}$$
$$a_2 = \langle x, \tilde{\varphi}_2 \rangle = \left\langle (3,1), (-\tfrac{1}{2},\tfrac{1}{2}) \right\rangle = -\tfrac{3}{2} + \tfrac{1}{2} = -1.$$

Support of a Function

- The **support** of a function f, denoted $\operatorname{supp} f$, is the closure of the set $\{t : f(t) \neq 0\}$ (i.e., the smallest closed set that contains all of the points where f is nonzero).
  Example. $\operatorname{supp} \operatorname{rect} = [-\frac{1}{2}, \frac{1}{2}]$ and $\operatorname{supp} \operatorname{sinc} = \mathbb{R}$.
- A function f defined on $\mathbb{R}$ is said to have **compact support** if $\operatorname{supp} f \subset [a,b]$ for $a,b \in \mathbb{R}$. (Note: The terms “finite duration” and “time limited” are synonymous with compact support.)
  Example. The rect function has compact support, while the sinc function does not have compact support.

Moments

- The kth **moment** of a sequence x defined on $\mathbb{Z}$ is given by $m_k = \sum_{n\in\mathbb{Z}} n^k x[n]$ (i.e., $m_k = \langle x, (\cdot)^k \rangle$).
- The kth moment of a function x defined on $\mathbb{R}$ is given by $m_k = \int_{-\infty}^{\infty} t^k x(t)dt$ (i.e., $m_k = \langle x, (\cdot)^k \rangle$).
- Moments are essentially inner products with monomials. Since monomials/polynomials play an important role in many contexts, moments are often of interest.
- A function or sequence f is said to have p **vanishing moments** if its first p moments vanish (i.e., $m_0 = m_1 = \ldots = m_{p-1} = 0$, where m_k is the kth moment of f).

Section 4.2

Best Basis for Signal Representation

Bases of Function Spaces

- for function x, often employ expansion of form

  $$x(t) = \sum_n a_n \varphi_n(t)$$

  expansion coefficients a_n, basis functions φ_n
- in practice, always use structured bases
- basis functions related to each other (e.g., by dilation, translation, modulation)
- if not structured: cumbersome to employ, computationally intractable algorithms
- may want basis functions to be localized in time and/or frequency, continuous, differentiable, smooth, have certain moment properties
- often desired properties can be in conflict with one another and compromises must be made

Fourier Basis for Space of Periodic Functions

- Consider the space V of all ***periodic functions*** f with fundamental period T and fundamental frequency $\omega_0 = \frac{2\pi}{T}$ such that $\int_{-T/2}^{T/2} |f(t)|^2 dt < \infty$.
- Any function $x \in V$ can be written as

$$x(t) = \sum_n a_n e^{jn\omega_0 t} \quad \text{where} \quad a_n = \frac{1}{T} \int_{-T/2}^{T/2} x(t) e^{-jn\omega_0 t} dt.$$

- The above is a basis representation of x in terms of the ***basis functions*** $\{\varphi_n\}_{n \in \mathbb{Z}}$, where

$$\varphi_n(t) = e^{jn\omega_0 t}.$$

- All of the basis functions $\{\varphi_n\}_{n \in \mathbb{Z}}$ are generated from a single prototype function $\varphi(t) = e^{j\omega_0 t}$ by ***dilation***. That is, $\varphi_n(t) = \varphi(nt)$.
- The above basis representation of $x(t)$ is known as a ***Fourier series***.
- The Fourier series has the disadvantage of slow convergence in the vicinity of discontinuities (i.e., Gibbs phenomenon).

Sinc Basis for Space of Bandlimited Functions

- Consider the space V comprised of all functions in $L^2(\mathbb{R})$ (i.e., finite-energy functions defined on $\mathbb{R}$) that are ***bandlimited*** to frequencies in the range $(-\omega_b, \omega_b)$.
- From the ***sampling theorem***, any function $x \in V$ can be represented as

$$x(t) = \sum_{n \in \mathbb{Z}} a[n] \operatorname{sinc}(\omega_b t - \pi n) \quad \text{where} \quad a[n] = x\left(\tfrac{\pi}{\omega_b} n\right).$$

  (Note that the sequence a is formed by ***periodic sampling*** of x.)
- The above is a basis representation of the function x in terms of the ***basis functions*** $\{\varphi_n\}_{n \in \mathbb{Z}}$, where

$$\varphi_n(t) = \operatorname{sinc}(\omega_b t - \pi n).$$

- The basis functions φ_n are all generated from a single prototype function $\varphi(t) = \operatorname{sinc}(\omega_b t)$ by ***translation***. That is,

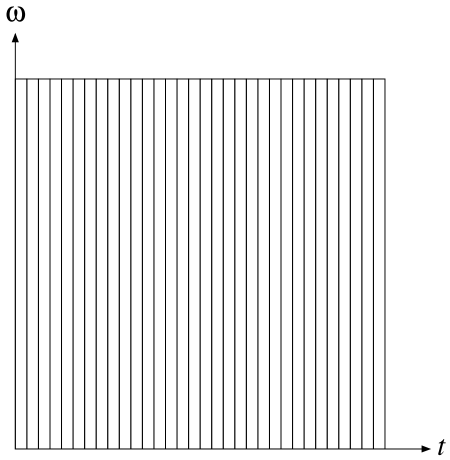
$$\varphi_n(t) = \varphi\left(t - \tfrac{\pi}{\omega_b} n\right).$$

Time-Frequency Tradeoff

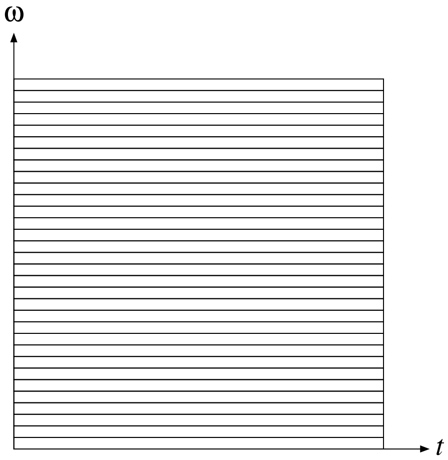
- ideally, would like to choose basis functions to be both bandlimited and time limited
- function cannot be both bandlimited and time limited
- cannot construct representation that simultaneously provides exact frequency and time resolution
- best we can do is choose basis functions so that most energy concentrated in finite interval in time and frequency
- can think of basis functions as creating tiling of time-frequency plane
- different choice of basis functions leads to different tilings

Time-Frequency Perspective on Signal Representations

- each basis function has particular energy distribution with respect to time and frequency
- with each basis function can associate a region (i.e., tile) in time-frequency plane where energy of function is mostly concentrated



Standard Basis



Fourier Basis

Appropriate Choice of Basis

- no single “best” basis
- which basis most appropriate depends on application at hand
- may require basis functions with good time and frequency resolution
- locating transient events such as signal singularities in time domain
- signal with many discontinuities might be better represented by basis functions with discontinuities
- self-similar signals might be better represented by basis functions having similar shape

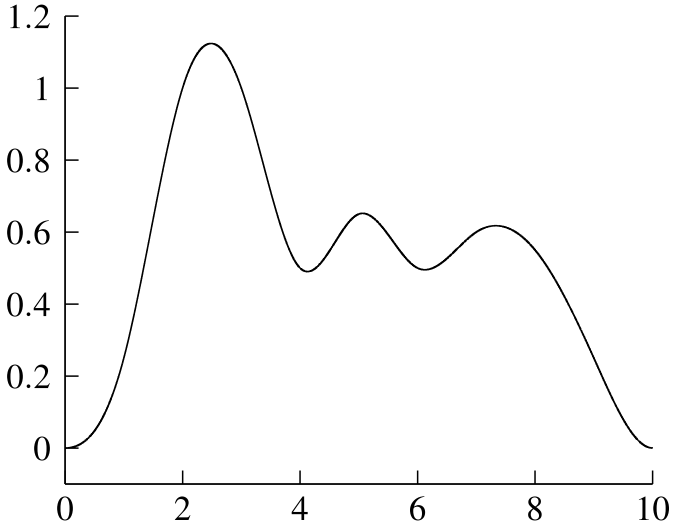
Section 4.3

Multiresolution Representations

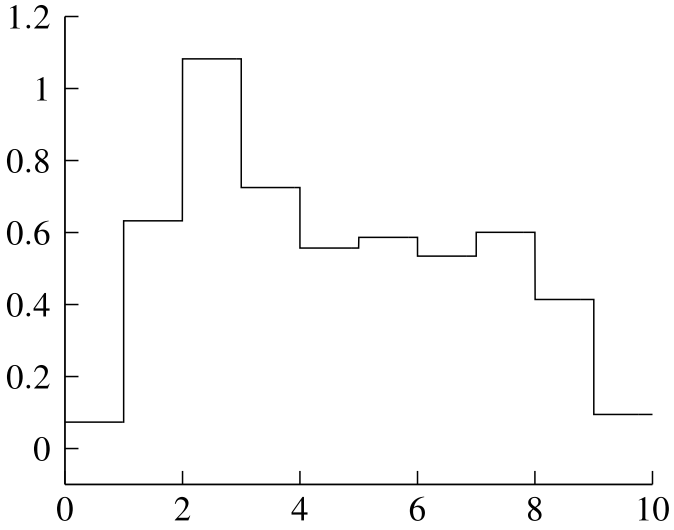
Multiresolution Representations

- Often, it is beneficial to be able to approximate a function at different resolutions (or scales).
- In the case of a low resolution (i.e., coarse scale) approximation of a function, many details are lost, but the general trends in function behavior are still apparent.
- In the case of a high resolution (i.e., fine scale) approximation of a function, more details of the original function are present.
- Typically, the use of higher resolution representations requires more computation and memory but yields more precise results.
- So, in practice, there is usually a fundamental tradeoff between computation/memory and precision of results.
- Multiresolution representations allow for the level of detail used in computations to be more easily adjusted to suit the needs of the application at hand.

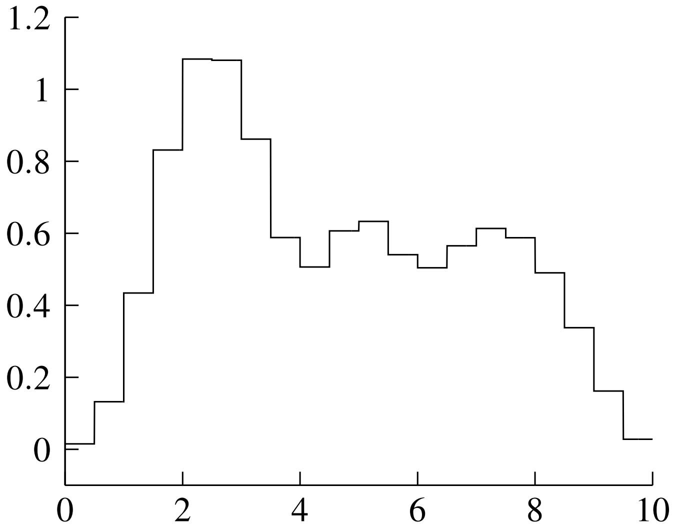
Multiresolution Approximation of Function



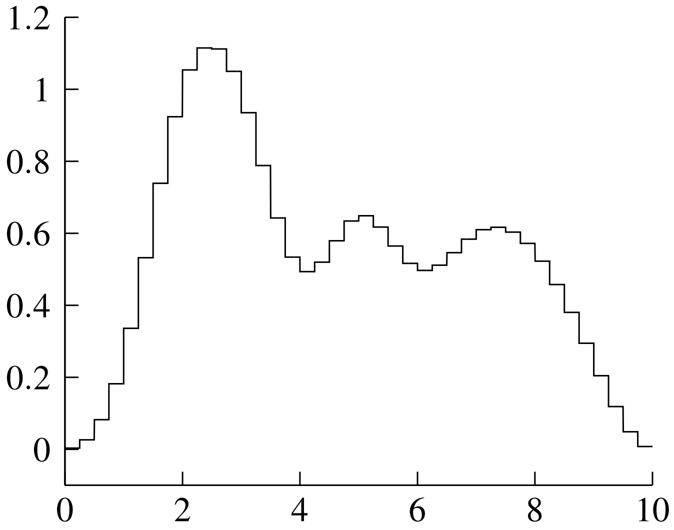
Original



Coarse Approximation

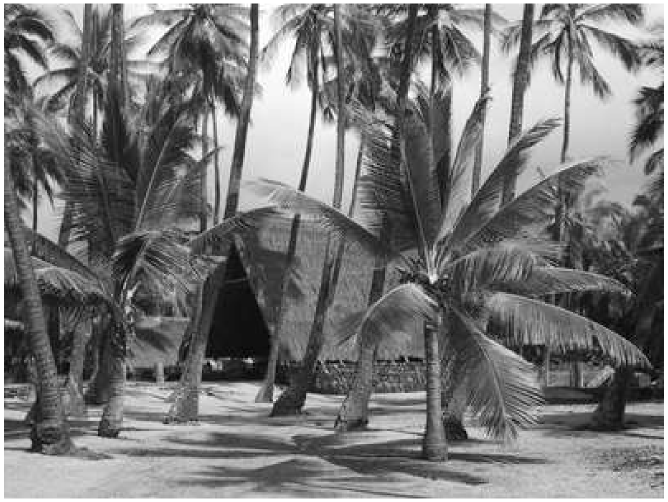


Medium Approximation

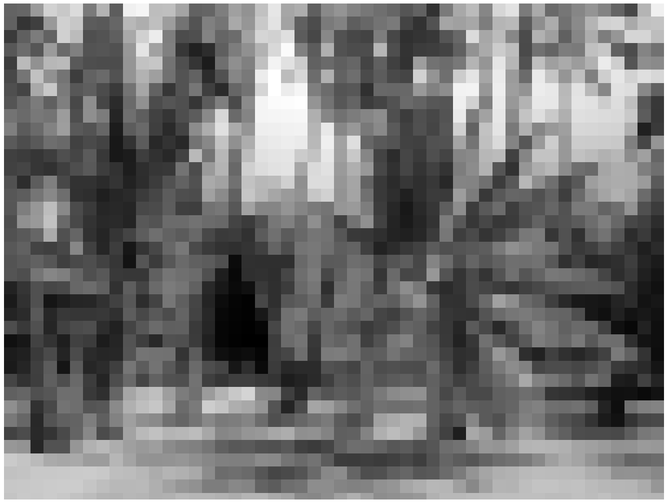


Fine Approximation

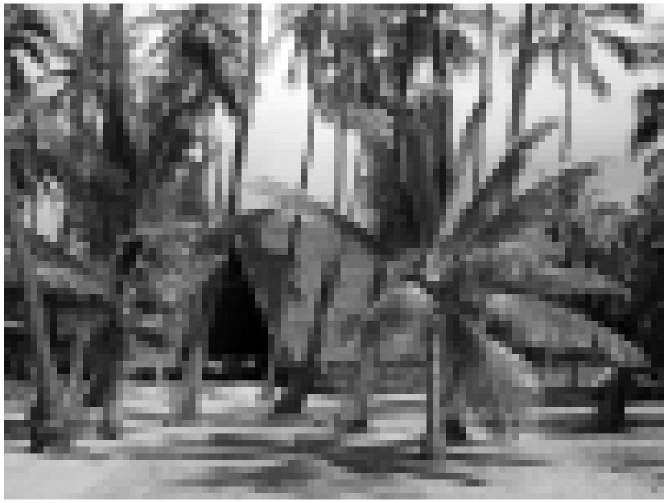
Multiresolution Approximation of 2-D Function



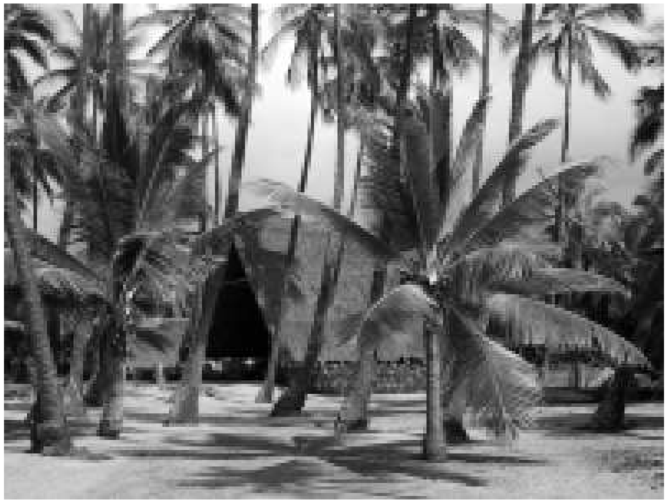
Original



Coarse Approximation



Medium Approximation



Fine Approximation

Dyadic Wavelet Systems

- Wavelet systems are a form of multiresolution representation where each successive resolution differs in scale by some ***integer factor***.
- In the case that this factor is two, we have what is called a **dyadic wavelet system**.
- Dyadic wavelet systems are arguably the most commonly-used type of wavelet system.
- Herein, we focus our attention on dyadic wavelet systems.

Section 4.4

Approximation and Wavelet Spaces

Approximation and Wavelet Spaces

- Wavelet systems are associated with a collection of function spaces having a particular structure.
- In what follows, the structure of these function spaces will be introduced.

Multiresolution Approximations (MRAs)

- A sequence $\{V_p\}_{p\in\mathbb{Z}}$ of closed subspaces of $L^2(\mathbb{R})$ is said to be a **multiresolution approximation** (MRA) if the following properties hold:
  1. for all $p \in \mathbb{Z}$, $V_p \subset V_{p-1}$ (**nesting**);
  2. $\lim\limits_{p\to\infty} V_p = \bigcap\limits_{p\in\mathbb{Z}} V_p = \{0\}$ (**separation**);
  3. $\lim\limits_{p\to-\infty} V_p = \operatorname{clos}\left(\bigcup\limits_{p\in\mathbb{Z}} V_p\right) = L^2(\mathbb{R})$ (**density**);
  4. for all $p \in \mathbb{Z}$, $f(t) \in V_p \Leftrightarrow f(2t) \in V_{p-1}$ (**scale invariance**);
  5. for all $k \in \mathbb{Z}$, $f(t) \in V_0 \Leftrightarrow f(t-k) \in V_0$ (**shift invariance**); and
  6. there exists ϕ such that $\{\phi(\cdot - n)\}_{n\in\mathbb{Z}}$ is a Riesz basis of V_0 (**shift-invariant Riesz basis**).
- The spaces V_p in the above definition are referred to as **approximation spaces** (or **scaling spaces**).
- A (Riesz) basis of V_0 is generated from the ***integer translates*** of a single prototype function ϕ.
- This prototype function is referred to as a **scaling function**.

MRAs (Continued)

- Some of the properties of a MRA given in the definition of MRA are ***redundant*** (i.e., some properties can be inferred from the others).
- The shift-invariance and scale-invariance properties together can be shown to imply that

$$\text{for all } p,k \in \mathbb{Z},\ \ f(t) \in V_p \Leftrightarrow f(t-2^p k) \in V_p.$$

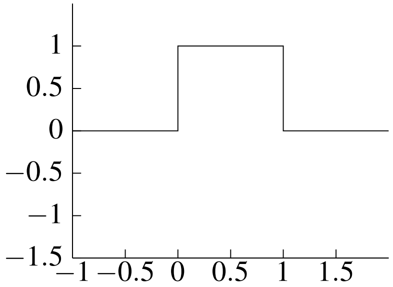
- The scale invariance property is fundamentally important, as it states that the functions in any two approximation spaces are identical, except for the ***horizontal scale***.
- Ultimately, the scale-invariance property leads to the inherent multiresolution nature of wavelet systems.
- As the index p increases, the scale of the approximation space V_p becomes coarser (i.e., the functions in V_p become increasingly stretched horizontally so as to change more slowly and therefore contain less high-frequency detail).

Piecewise Constant Approximations

- The simplest MRA is associated with ***piecewise constant*** approximations.
- The space V_p is comprised of all functions $f \in L^2(\mathbb{R})$ such that f is ***constant*** on intervals of the form $[n2^p, (n+1)2^p)$, where $n \in \mathbb{Z}$.
- For example, V_0 is comprised of all functions $f \in L^2(\mathbb{R})$ such that f is ***constant*** on intervals of the form $[n, n+1)$, where $n \in \mathbb{Z}$.
- One can show that an orthonormal basis of V_0 is given by $\{\phi(\cdot - k)\}_{k \in \mathbb{Z}}$, where

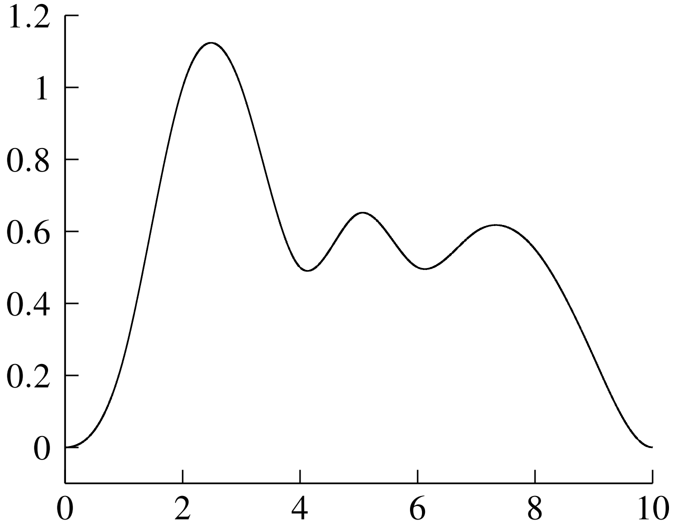
$$\phi(t) = \chi_{[0,1)}(t).$$

- A plot of ϕ is given below.

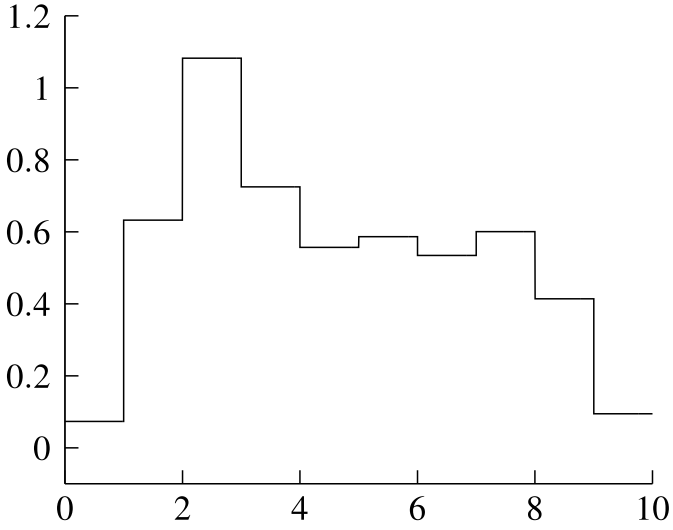


- Piecewise constant approximations are ***not continuous***, which can be undesirable in some applications (e.g., when smooth approximations of smooth functions are needed).

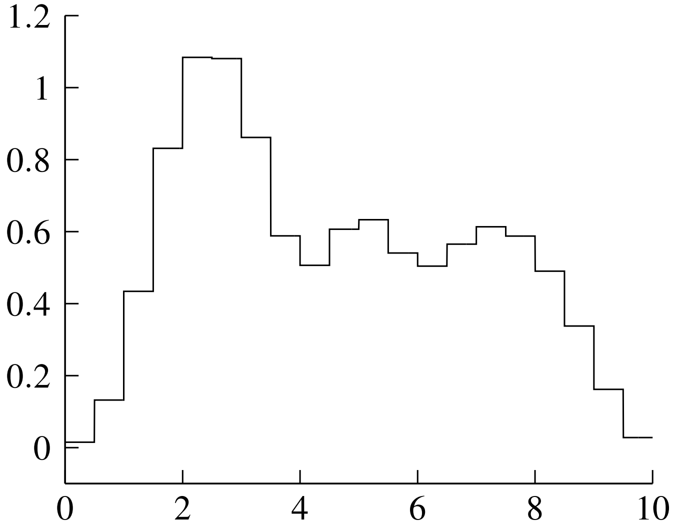
Piecewise Constant Approximations



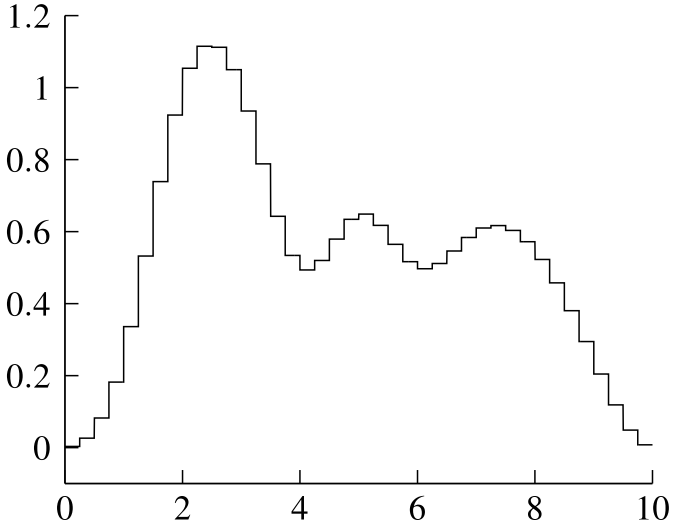
Original Function



Projection onto V_0



Projection onto V_{-1}



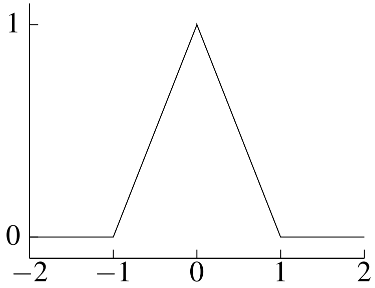
Projection onto V_{-2}

Continuous Piecewise-Linear Approximations

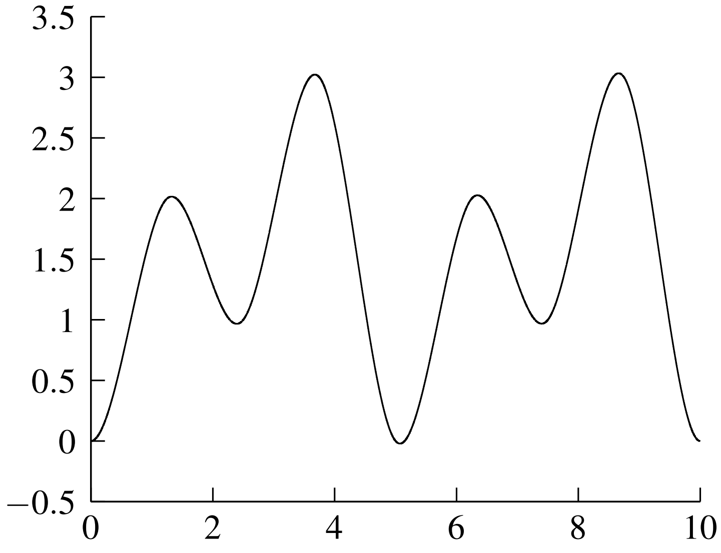
- Consider a MRA associated with ***continuous piecewise-linear*** approximations.
- The space V_p is the set of all ***continuous*** functions $f \in L^2(\mathbb{R})$ such that f is ***linear*** on intervals of the form $[n2^p, (n+1)2^p)$.
- For example, the space V_0 is the set of all ***continuous*** functions $f \in L^2(\mathbb{R})$ such that f is ***linear*** on intervals of the form $[n, n+1)$, where $n \in \mathbb{Z}$.
- One can show that a (Riesz) basis of V_0 is given by $\{\phi(\cdot - k)\}_{k \in \mathbb{Z}}$, where

$$\phi(t) = \begin{cases} 1 - |t| & \text{for } t \in [-1, 1] \\ 0 & \text{otherwise.} \end{cases}$$

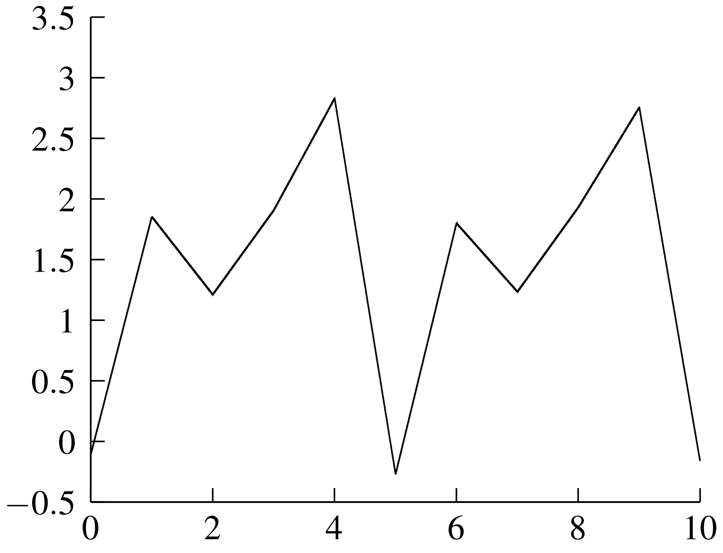
  (The above basis is not orthonormal, however.)
- A plot of ϕ is given below.



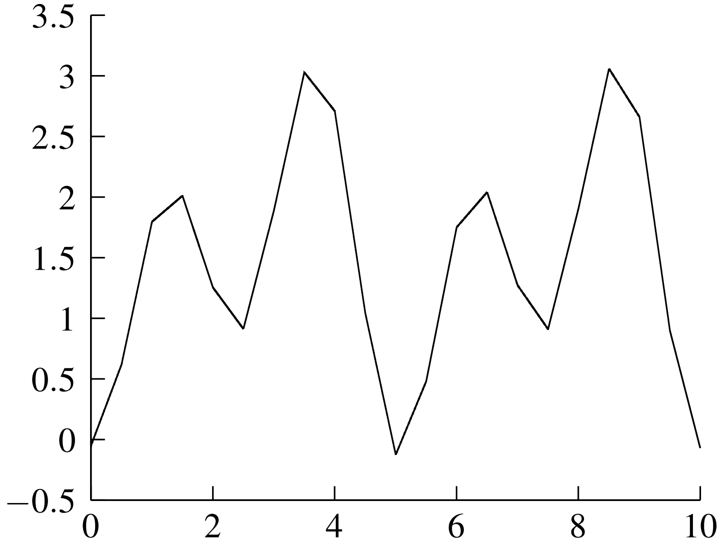
Continuous Piecewise Linear Approximations



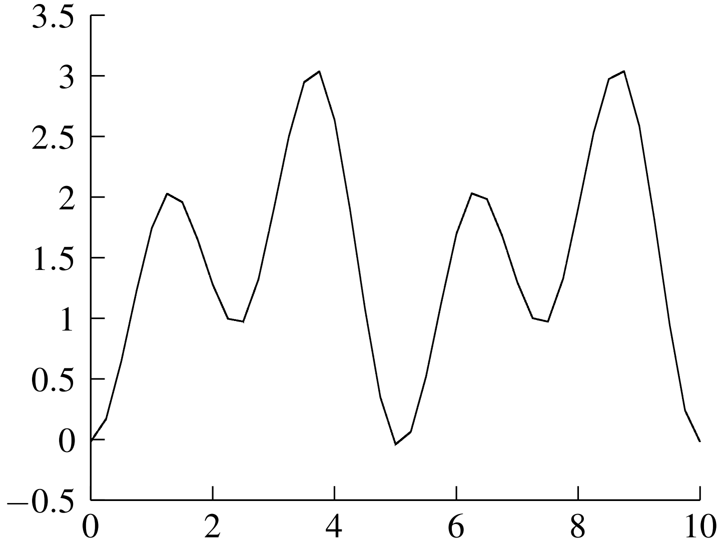
Original Function



Projection onto V_0



Projection onto V_{-1}



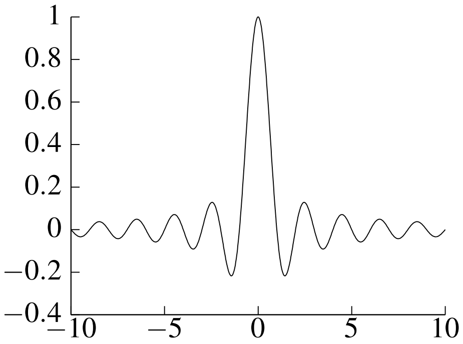
Projection onto V_{-2}

Shannon Approximations

- The Shannon approximation utilizes ***bandlimited*** functions in order to form its approximation spaces.
- The space V_p is comprised of the set of all $f \in L^2(\mathbb{R})$ such that $\operatorname{supp} \hat{f} \subset [-2^{-p}\pi, 2^{-p}\pi]$ (i.e., f is ***bandlimited*** to frequencies in the range $-2^{-p}\pi$ to $2^{-p}\pi$).
- For example, the space V_0 is the set of all $f \in L^2(\mathbb{R})$ such that $\operatorname{supp} \hat{f} \subset [-\pi, \pi]$ (i.e., f is ***bandlimited*** to frequencies in the range $[-\pi, \pi]$).
- One can show that an orthonormal basis of V_0 is given by $\{\phi(\cdot - k)\}_{k \in \mathbb{Z}}$, where

$$\phi(t) = \operatorname{sinc} \pi t.$$

- A plot of ϕ is shown below.



Wavelet Spaces

- As it turns out, the approximation spaces of a MRA are not the only spaces of interest in the context of wavelet systems.
- Consider a MRA associated with the approximation space sequence $\{V_p\}_{p \in \mathbb{Z}}$.
- For each $p \in \mathbb{Z}$, since V_p is a proper subspace of V_{p-1} (i.e., $V_p \subset V_{p-1}$ and $V_p \neq V_{p-1}$), there must exist some space W_p such that W_p is the ***algebraic complement*** of V_p in V_{p-1}.
- That is, there must exist a space W_p such that
$$V_{p-1} = V_p \oplus W_p.$$
- The space W_p is referred to as a **wavelet space**.
- Thus, we can associate a sequence $\{W_p\}_{p \in \mathbb{Z}}$ of wavelet spaces with a MRA.
- From the structure of a MRA and the definition of the wavelet spaces, it can be shown that
$$\begin{gathered} V_k \cap V_l = V_l \quad \text{for } k < l, \\ W_k \cap W_l = \{0\} \quad \text{for } k \neq l, \quad \text{and} \\ V_k \cap W_l = \{0\} \quad \text{for } k \geq l. \end{gathered}$$

Decomposition of $L^2(\mathbb{R})$ Using Wavelet Spaces

- Using the fact that $V_{p-1} = V_p \oplus W_p$, one can show that

$$\operatorname{clos}\left(\bigoplus_{p\in\mathbb{Z}} W_p\right) = L^2(\mathbb{R}).$$

- In other words, the space $L^2(\mathbb{R})$ can be decomposed into a sequence of ***mutually disjoint*** subspaces by using the wavelet spaces $\{W_p\}_{p\in\mathbb{Z}}$.
- By combining the bases for the wavelet spaces $\{W_p\}_{p\in\mathbb{Z}}$, we can obtain a ***Riesz basis*** for $L^2(\mathbb{R})$.
- This particular structure leads to biorthogonal wavelet systems.
- Diagrammatically, we have decomposed $L^2(\mathbb{R})$ using the structure illustrated below.

$$\begin{tikzcd}
L^2(\mathbb{R}) \arrow[r] & \cdots V_{-2} \arrow[r] \arrow[dr] & V_{-1} \arrow[r] \arrow[dr] & V_0 \arrow[r] \arrow[dr] & V_1 \arrow[r] \arrow[dr] & V_2 \cdots \arrow[r] & \{0\} \\
& & W_{-1} & W_0 & W_1 & W_2 &
\end{tikzcd}$$

Decomposition of $L^2(\mathbb{R})$ Using Wavelet Spaces (Continued)

- Suppose now that, in addition to $V_{p-1} = V_p \oplus W_p$, we have that $V_p \perp W_p$. Then, one can show
$$\operatorname{clos}\left(\bigoplus_{p\in\mathbb{Z}}^{\perp} W_p\right) = L^2(\mathbb{R}).$$
- In other words, the space $L^2(\mathbb{R})$ can be decomposed into a sequence of ***mutually orthogonal*** subspaces using the wavelet spaces $\{W_p\}_{p\in\mathbb{Z}}$.
- By combining orthonormal/Riesz bases for the wavelet spaces $\{W_p\}_{p\in\mathbb{Z}}$, we can obtain an ***orthonormal/Riesz basis*** for $L^2(\mathbb{R})$.
- This particular structure leads to semiorthogonal and orthonormal wavelet systems.

Properties of Wavelet Spaces

- First, as noted earlier, the wavelet spaces $\{W_p\}_{p\in\mathbb{Z}}$ are ***mutually disjoint***.
- **Theorem.** Let $\{W_p\}_{p\in\mathbb{Z}}$ denote the sequence of wavelet spaces associated with a MRA. The wavelet spaces are such that
  1. $\operatorname{clos}\left(\bigoplus_{p\in\mathbb{Z}} W_p\right) = L^2(\mathbb{R})$ (**density**);
  2. for all $p\in\mathbb{Z}$, $f(t)\in W_p \Leftrightarrow f(2t)\in W_{p-1}$ (**scale invariance**);
  3. for all $k\in\mathbb{Z}$, $f(t)\in W_0 \Leftrightarrow f(t-k)\in W_0$ (**shift invariance**); and
  4. there exists ψ such that $\{\psi(\cdot-k)\}_{k\in\mathbb{Z}}$ is a Riesz basis of W_0 (**shift-invariant Riesz basis**).
- From the above theorem, we see that a Riesz basis of the wavelet space W_0 is generated by the integer translates of a single prototype function ψ.
- This prototype function ψ is called a **wavelet function**.
- Essentially, the wavelet function plays the same role for wavelet spaces as the scaling function does for approximation spaces.

Practical Comment on Approximation and Wavelet Spaces

- In practice, we often have a function f represented in some approximation space, say V_ρ, and we want to find a representation of f in terms of the approximation space $V_{\rho'}$, where $\rho' > \rho$, and wavelet spaces W_p for $\rho < p \leq \rho'$.
- That is, we want to express the function f as the sum of its lower resolution representation in $V_{\rho'}$ and the additional details in $\{W_p\}_{p=\rho+1}^{\rho'}$ necessary to obtain the original function f.
- In such a situation, we employ a decomposition of the form

$$\begin{aligned} V_\rho &= V_{\rho+1} \oplus W_{\rho+1} \\ &= V_{\rho+2} \oplus W_{\rho+2} \oplus W_{\rho+1} \\ &= V_{\rho'} \oplus \bigoplus_{k=\rho+1}^{\rho'} W_k. \end{aligned}$$

- That is, we can express $f \in V_\rho$ as $f = f_{V_{\rho'}} + f_{W_{\rho+1}} + f_{W_{\rho+2}} + \ldots + f_{W_{\rho'}}$, where f_X denotes a function in the space X.
- Notice that we are only using a ***finite*** number of approximation and wavelet spaces.

Bases of Approximation Spaces

- **Theorem.** Suppose that we have a MRA $\{V_p\}_{p\in\mathbb{Z}}$ and $\{\phi(\cdot - n)\}_{n\in\mathbb{Z}}$ is a Riesz basis of V_0 with the dual basis $\{\tilde{\phi}(\cdot - n)\}_{n\in\mathbb{Z}}$. Then, for each $p \in \mathbb{Z}$, $\{\phi_{p,k}\}_{k\in\mathbb{Z}}$ given by

  $$\phi_{p,k}(t) = 2^{-p/2}\phi(2^{-p}t - k)$$

  is a Riesz basis of V_p with the dual basis $\{\tilde{\phi}_{p,k}\}_{k\in\mathbb{Z}}$ given by

  $$\tilde{\phi}_{p,k}(t) = 2^{-p/2}\tilde{\phi}(2^{-p}t - k).$$

- In other words, the basis functions $\{\phi_{p,k}\}_{k\in\mathbb{Z}}$ for each approximation space V_p are ***translated***, ***dilated***, and ***scaled*** versions of the scaling function ϕ.

Bases of Wavelet Spaces

- **Theorem.** Suppose that we have a MRA $\{V_p\}_{p\in\mathbb{Z}}$ with corresponding wavelet spaces $\{W_p\}_{p\in\mathbb{Z}}$ (i.e., $V_{p-1} = V_p \oplus W_p$) and $\{\psi(\cdot - n)\}_{n\in\mathbb{Z}}$ is a Riesz basis of W_0 with the dual basis $\{\tilde{\psi}(\cdot - n)\}_{n\in\mathbb{Z}}$. Then, for each $p \in \mathbb{Z}$, $\{\psi_{p,k}\}_{k\in\mathbb{Z}}$ given by
$$\psi_{p,k}(t) = 2^{-p/2}\psi(2^{-p}t - k)$$
is a Riesz basis of W_p with the dual basis $\{\tilde{\psi}_{p,k}\}_{k\in\mathbb{Z}}$ given by
$$\tilde{\psi}_{p,k}(t) = 2^{-p/2}\tilde{\psi}(2^{-p}t - k).$$

- In other words, the basis functions $\{\psi_{p,k}\}_{k\in\mathbb{Z}}$ for each wavelet space W_p are ***translated***, ***dilated***, and ***scaled*** versions of the wavelet function ψ.

Section 4.5

Scaling and Wavelet Equations

Refinement Equations

- An equation of the form

  $$\phi(t) = \sum_{n\in\mathbb{Z}} a[n]\phi(2t-n)$$

  is called a **refinement equation** (or **dilation equation**).
- The sequence a is referred to as a **refinement mask**.
- A solution ϕ of the above refinement equations is called a **refinable function**.
- Clearly, any dilation equation admits a ***trivial solution*** of $\phi(t) \equiv 0$.
- If a nontrivial solution does exist, the solution is ***not unique***.
- If $\phi(t)$ is a solution, then $\alpha\phi(t)$ is also a solution, where α is a constant.
- Although some refinable functions can be expressed in closed form, most cannot be expressed in this way.

Haar Scaling Function

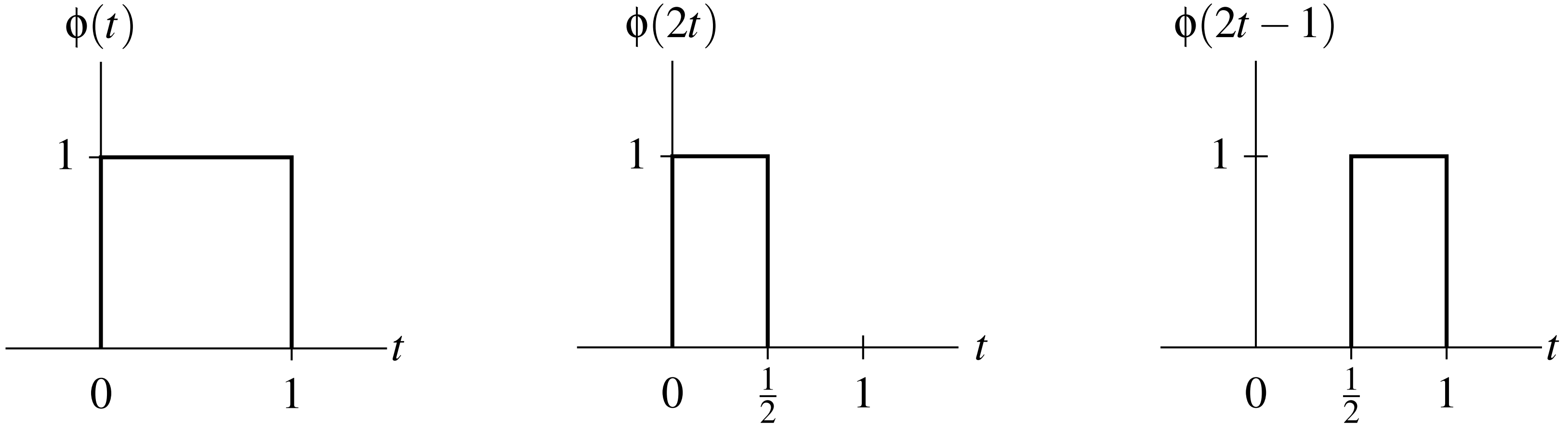
- Consider the function

$$\phi(t) = \chi_{[0,1)}(t).$$

- One can confirm that ϕ satisfies the ***refinement equation***

$$\phi(t) = \phi(2t) + \phi(2t-1).$$

- This refinement relationship is illustrated below.



Linear B-Spline Scaling Function

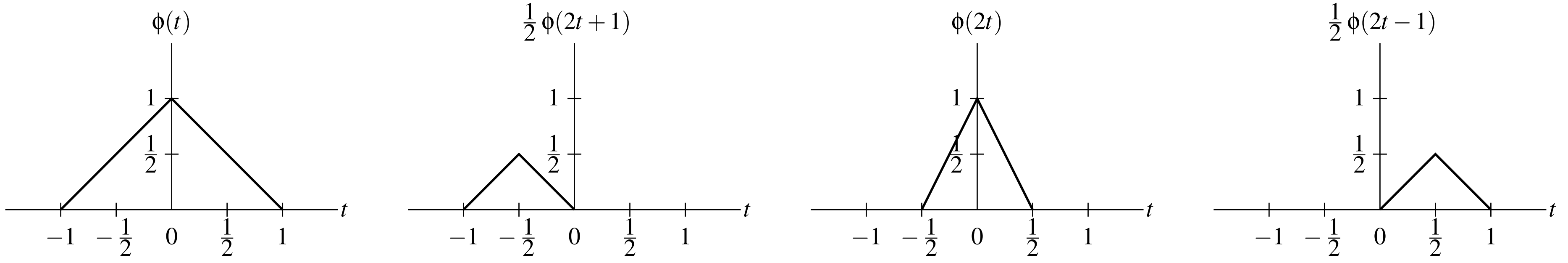
- Consider the function

$$\phi(t) = \begin{cases} 1 - |t| & \text{for } t \in [-1,1] \\ 0 & \text{otherwise.} \end{cases}$$

- One can confirm that ϕ satisfies the ***refinement equation***

$$\phi(t) = \tfrac{1}{2}\phi(2t+1) + \phi(2t) + \tfrac{1}{2}\phi(2t-1).$$

- This refinement relationship is illustrated below.



Scaling Equation

- As it turns out, the scaling function ϕ of a MRA satisfies a refinement equation (i.e., ϕ is ***refinable***).
- **Theorem.** Suppose that we have a MRA $\{V_p\}_{p\in\mathbb{Z}}$ and V_0 has the Riesz basis $\{\phi(\cdot - n)\}_{n\in\mathbb{Z}}$. Then, ϕ satisfies a ***refinement equation*** of the form

$$\phi(t) = \sqrt{2}\sum_{n\in\mathbb{Z}} c[n]\phi(2t-n),$$

  where

$$c[n] = \left\langle \phi(\cdot), \sqrt{2}\tilde{\phi}(2\cdot - n)\right\rangle.$$

- The above refinement equation is referred to as a ***scaling equation***.
- Due to the above relationship, refinement equations play an important role in the study of wavelet systems.
- By convention, the scaling equation is written so as to contain an explicit $\sqrt{2}$ factor. Hence, the refinement mask is actually $\sqrt{2}c$ (and not c).

Fourier Transform of Scaling Equation

- **Theorem.** Let ϕ be a scaling function with scaling equation coefficient sequence c (i.e., $\phi(t) = \sqrt{2}\sum_{n\in\mathbb{Z}} c[n]\phi(2t-n)$). Then, $\hat{\phi}$ is given by
$$\hat{\phi}(\omega) = \tfrac{1}{\sqrt{2}}\hat{c}(\tfrac{\omega}{2})\hat{\phi}(\tfrac{\omega}{2}),$$
which can be equivalently expressed in terms of an infinite product as
$$\hat{\phi}(\omega) = \hat{\phi}(0)\prod_{p=1}^{\infty}\frac{\hat{c}(2^{-p}\omega)}{\sqrt{2}}.$$

Wavelet Equation

- Although the wavelet function does not satisfy a refinement equation (i.e., is ***not refinable***), it can be expressed in terms of dilated and translated versions of the scaling function.
- **Theorem.** Suppose that we have a MRA $\{V_p\}_{p\in\mathbb{Z}}$ with wavelet subspaces $\{W_p\}_{p\in\mathbb{Z}}$, where V_0 has the Riesz basis $\{\phi(\cdot-n)\}_{n\in\mathbb{Z}}$ and W_0 has the Riesz basis $\{\psi(\cdot-n)\}_{n\in\mathbb{Z}}$. Then, $\psi(t)$ can be expressed in terms of an equation of the form

  $$\psi(t) = \sqrt{2}\sum_{n\in\mathbb{Z}} d[n]\phi(2t-n)$$

  where

  $$d[n] = \left\langle \psi(\cdot), \sqrt{2}\tilde{\phi}(2\cdot-n)\right\rangle.$$

- The above equation for ψ is referred to as a ***wavelet equation***.
- Note that the wavelet equation is ***not*** a refinement equation.

Fourier Transform of Wavelet Equation

- **Theorem.** Let ϕ and ψ be the scaling and wavelet functions of a MRA. Suppose that ϕ has scaling equation coefficient sequence c and ψ has the wavelet equation coefficient sequence d. Then, $\hat{\psi}$ is given by

$$\hat{\psi}(\omega) = \tfrac{1}{\sqrt{2}}\hat{d}(\tfrac{\omega}{2})\hat{\phi}(\tfrac{\omega}{2}),$$

which can be equivalently expressed in terms of an infinite product as

$$\hat{\psi}(\omega) = \tfrac{1}{\sqrt{2}}\hat{\phi}(0)\hat{d}(\tfrac{\omega}{2})\prod_{p=1}^{\infty}\tfrac{1}{\sqrt{2}}\hat{c}(2^{-p-1}\omega).$$

Generating MRAs from Scaling Functions

- Rather than defining an MRA in terms of its approximation spaces, we can use the scaling function as the starting point for defining a MRA.
- To begin, we make an appropriate choice of scaling function ϕ.
- Then, from this scaling function, we can generate each of the approximation spaces of the MRA.
- This process is formalized by the theorem below.
- **Theorem.** Suppose that $\phi \in L^2(\mathbb{R})$ satisfies a refinement relation of the form

$$\phi(t) = \sum_{k \in \mathbb{Z}} c[k]\phi(2t-k),$$

where $\sum_{k \in \mathbb{Z}} |c[k]|^2 < \infty$. Further, suppose that $\{\phi(\cdot - k)\}_{k \in \mathbb{Z}}$ is a Riesz basis for the space that it generates. Define

$$\phi_{p,k}(t) = 2^{-p/2}\phi(2^{-p}t-k)$$

and let V_p be the space generated by $\{\phi_{p,k}\}_{k \in \mathbb{Z}}$. Then, the sequence $\{V_p\}_{p \in \mathbb{Z}}$ of spaces constitutes a MRA.

Dual MRAs

- One might wonder what structure (if any) is associated with the functions $\tilde{\phi}$ and $\tilde{\psi}$.
- **Theorem.** Let $\{V_p\}_{p\in\mathbb{Z}}$ be a MRA with scaling function ϕ, wavelet space sequence $\{W_p\}_{p\in\mathbb{Z}}$, and corresponding wavelet function ψ. Suppose that the dual Riesz bases of $\{\phi(\cdot - k)\}_{k\in\mathbb{Z}}$ and $\{\psi(\cdot - k)\}_{k\in\mathbb{Z}}$ are given by $\{\tilde{\phi}(\cdot - k)\}_{k\in\mathbb{Z}}$ and $\{\tilde{\psi}(\cdot - k)\}_{k\in\mathbb{Z}}$, respectively. Then, $\tilde{\phi}$ is also the scaling function of a MRA $\{\tilde{V}_p\}_{p\in\mathbb{Z}}$ with wavelet space sequence $\{\tilde{W}_p\}_{p\in\mathbb{Z}}$ and the corresponding wavelet function $\tilde{\psi}$.
- The above theorem is significant because it shows that MRAs ***occur in pairs***.
- The MRA $\{\tilde{V}_p\}_{p\in\mathbb{Z}}$ is said to be the **dual MRA** of $\{V_p\}_{p\in\mathbb{Z}}$.
- Furthermore, it follows that if $\{\tilde{V}_p\}_{p\in\mathbb{Z}}$ is the dual of $\{V_p\}_{p\in\mathbb{Z}}$ then $\{V_p\}_{p\in\mathbb{Z}}$ is also trivially the dual of $\{\tilde{V}_p\}_{p\in\mathbb{Z}}$. In other words, this duality property is symmetric.
- The functions $\tilde{\phi}$ and $\tilde{\psi}$ are referred to as the **dual scaling function** and **dual wavelet function**, respectively.
- The qualifier '***primal***" is used to mean "non-dual".

Dual MRAs

- **Theorem.** Suppose that $\{V_p\}_{p \in \mathbb{Z}}$ and $\{\tilde{V}_p\}_{p \in \mathbb{Z}}$ are dual MRAs with corresponding wavelet space sequences $\{W_p\}_{p \in \mathbb{Z}}$ and $\{\tilde{W}_p\}_{p \in \mathbb{Z}}$, respectively. Then, we have

$$\text{for all } p \in \mathbb{Z}, \quad V_p \perp \tilde{W}_p \quad \text{and} \quad W_p \perp \tilde{V}_p.$$

- In light of the above result, we observe that if the corresponding approximation and wavelet spaces associated with a MRA are orthogonal (i.e., for all $p \in \mathbb{Z}$, $V_p \perp W_p$), then the MRA is self dual.

Section 4.6

Wavelet Systems

Wavelet Systems

- A wavelet system is simply a ***basis of $L^2(\mathbb{R})$*** that is derived from a MRA.
- When constructing wavelet systems, we have a number of ***degrees of freedom*** available to us.
- In particular, we have some flexibility in the structure of approximation and wavelet spaces and the bases employed for these spaces.
- By exploiting this flexibility, we can obtain wavelet systems with differing types of structure.
- This leads to ***three different types*** of wavelet systems:
  1. orthonormal,
  2. semiorthogonal, and
  3. biorthonormal.

Orthonormal Wavelet Systems

- The most constrained type of wavelet system is an orthonormal wavelet system.
- With this type of system, the basis of each of the approximation and wavelet spaces is chosen to be ***orthonormal***, and each wavelet space is chosen to be ***orthogonal*** to its corresponding approximation space.
- That is, we have

$$\{\phi(\cdot - k)\}_{k \in \mathbb{Z}} \text{ is } \textit{orthonormal}, \quad \{\psi(\cdot - k)\}_{k \in \mathbb{Z}} \text{ is } \textit{orthonormal}, \quad \text{and}$$
$$\text{for each } p \in \mathbb{Z},\ V_p \perp W_p.$$

- From this, it follows that

$$\begin{gathered} \tilde{\phi} = \phi, \quad \tilde{\psi} = \psi, \quad \{\phi(\cdot - k)\}_{k \in \mathbb{Z}} \perp \{\psi(\cdot - k)\}_{k \in \mathbb{Z}}, \\ \tilde{V}_p = V_p, \quad \tilde{W}_p = W_p, \quad \{W_p\}_{p \in \mathbb{Z}} \text{ is mutually orthogonal.} \end{gathered}$$

- The MRA $\{V_p\}_{p \in \mathbb{Z}}$ is ***self dual***. That is, there is only one (distinct) MRA associated with an orthonormal wavelet system.

Semiorthogonal Wavelet Systems

- Sometimes, it can be overly restrictive to require the use of an orthonormal basis for each of the approximation and wavelet spaces.
- Dropping this constraint leads to what is called a semiorthogonal wavelet system.
- With this type of system, we choose to use a ***Riesz basis*** for each of the approximation and wavelet spaces, and each wavelet space is chosen to be ***orthogonal*** to its corresponding approximation space.
- That is, we have

$$\{\phi(\cdot - k)\}_{k\in\mathbb{Z}} \text{ is a } \textbf{\textit{Riesz basis}}, \quad \{\psi(\cdot - k)\}_{k\in\mathbb{Z}} \text{ is a } \textbf{\textit{Riesz basis}}, \quad \text{and}$$

$$\text{for each } p \in \mathbb{Z},\ V_p \perp W_p.$$

- From this, it follows that

$$\begin{gathered}\{\phi(\cdot - k)\}_{k\in\mathbb{Z}} \text{ and } \{\tilde{\phi}(\cdot - k)\}_{k\in\mathbb{Z}} \text{ are dual Riesz bases,}\\ \{\psi(\cdot - k)\}_{k\in\mathbb{Z}} \text{ and } \{\tilde{\psi}(\cdot - k)\}_{k\in\mathbb{Z}} \text{ are dual Riesz bases,}\\ \{\phi(\cdot - k)\}_{k\in\mathbb{Z}} \perp \{\psi(\cdot - k)\}_{k\in\mathbb{Z}}, \quad \{\tilde{\phi}(\cdot - k)\}_{k\in\mathbb{Z}} \perp \{\tilde{\psi}(\cdot - k)\}_{k\in\mathbb{Z}},\\ \tilde{V}_p = V_p, \quad \tilde{W}_p = W_p, \quad \text{and} \quad \{W_p\}_{p\in\mathbb{Z}} \text{ is mutually orthogonal.}\end{gathered}$$

- The MRA $\{V_p\}_{p\in\mathbb{Z}}$ is ***self dual***. Thus, there is only one (distinct) MRA associated with a semiorthogonal system.

Biorthogonal Wavelet Systems

- Sometimes, even the requirement that the corresponding approximation and wavelet spaces be orthogonal is too restrictive.
- By dropping this constraint, we obtain what is called a biorthonormal wavelet system.
- That is, we have

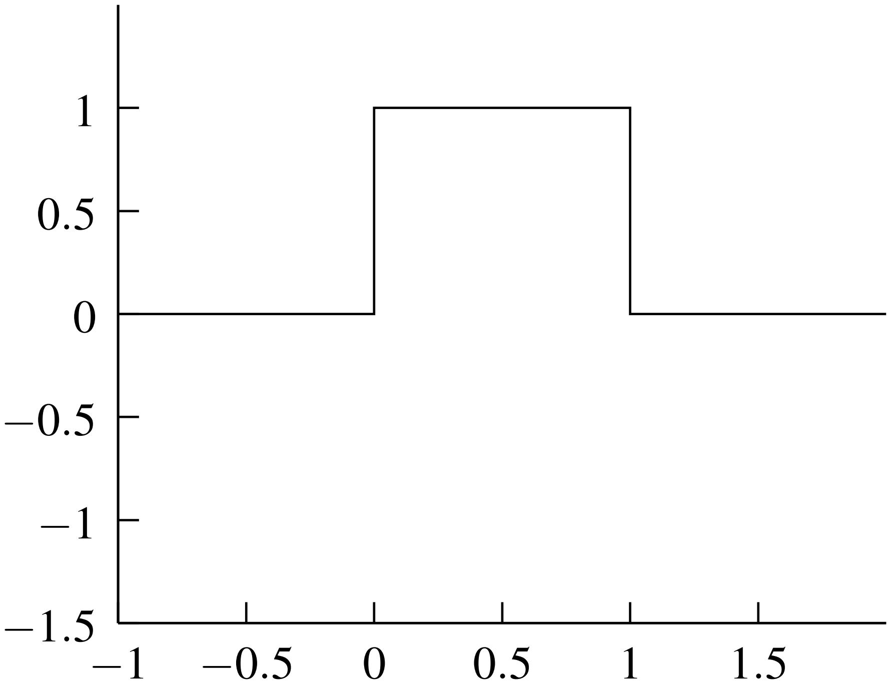
  $\{\phi(\cdot - k)\}_{k\in\mathbb{Z}}$ is a ***Riesz basis***, $\{\psi(\cdot - k)\}_{k\in\mathbb{Z}}$ is a ***Riesz basis***, and for each $p \in \mathbb{Z}$, $V_p \perp \tilde{W}_p$ and $W_p \perp \tilde{V}_p$.
- From this, it follows that

  $\{\phi(\cdot - k)\}_{k\in\mathbb{Z}}$ and $\{\tilde{\phi}(\cdot - k)\}_{k\in\mathbb{Z}}$ are dual Riesz bases,
  $\{\psi(\cdot - k)\}_{k\in\mathbb{Z}}$ and $\{\tilde{\psi}(\cdot - k)\}_{k\in\mathbb{Z}}$ are dual Riesz bases,
  $\{\phi(\cdot - k)\}_{k\in\mathbb{Z}} \perp \{\tilde{\psi}(\cdot - k)\}_{k\in\mathbb{Z}}, \quad \{\psi(\cdot - k)\}_{k\in\mathbb{Z}} \perp \{\tilde{\phi}(\cdot - k)\}_{k\in\mathbb{Z}},$
  $\{W_p\}_{p\in\mathbb{Z}}$ is mutually disjoint, and $\{\tilde{W}_p\}_{p\in\mathbb{Z}}$ is mutually disjoint.
- With a biorthonormal system, it is not necessarily true that $V_p \perp W_p$.
- So, $\{\phi(\cdot - k)\}_{k\in\mathbb{Z}}$ and $\{\psi(\cdot - k)\}_{k\in\mathbb{Z}}$ are not necessarily orthogonal.
- Moreover, the spaces in $\{W_p\}_{p\in\mathbb{Z}}$ are not necessarily mutually orthogonal. (These spaces are only guaranteed to be mutually disjoint.)
- The MRA $\{V_p\}_{p\in\mathbb{Z}}$ is no longer necessarily self dual. (Thus, there are potentially two distinct MRAs associated with a biorthonormal wavelet system.)

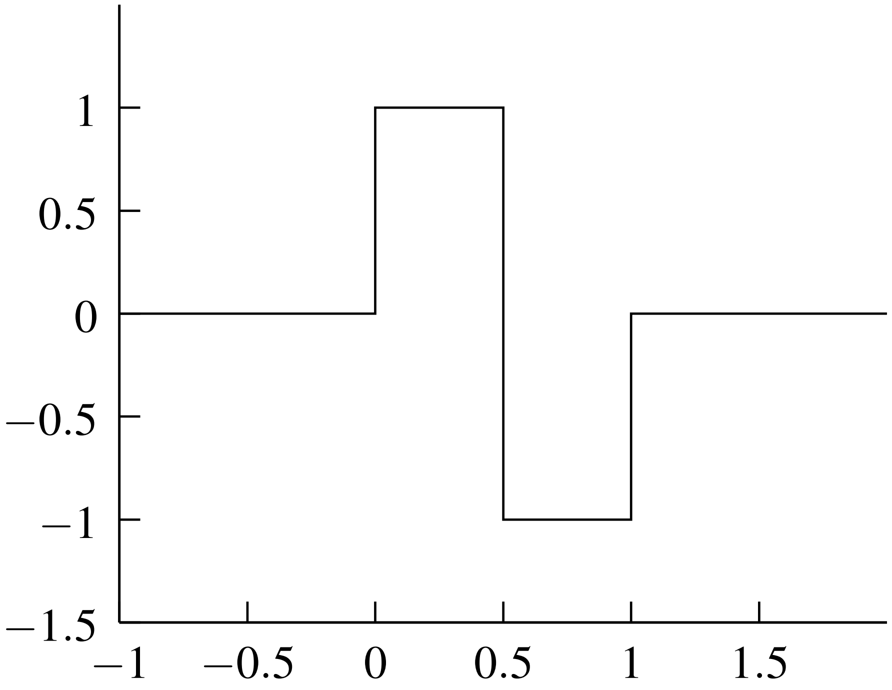
Haar Wavelet System

- One of the simplest examples of an orthonormal wavelet system is the Haar wavelet system.
- This system is associated with the MRA based on ***piecewise constant approximations*** (introduced earlier).
- The scaling function ϕ satisfies the refinement relationship
$$\phi(t) = \sqrt{2}\left(\tfrac{1}{\sqrt{2}}\phi(2t) + \tfrac{1}{\sqrt{2}}\phi(2t-1)\right).$$
- The wavelet function ψ can be expressed in terms of the scaling function as
$$\psi(t) = \sqrt{2}\left(\tfrac{1}{\sqrt{2}}\phi(2t) - \tfrac{1}{\sqrt{2}}\phi(2t-1)\right).$$
- Fortunately, ϕ and ψ can be expressed in closed form as
$$\phi(t) = \chi_{[0,1)}(t) \quad \text{and}$$
$$\psi(t) = \chi_{[0,1/2)}(t) - \chi_{[1/2,1)}(t) = \begin{cases} 1 & \text{for } t \in [0, \frac{1}{2}) \\ -1 & \text{for } t \in [\frac{1}{2}, 1) \\ 0 & \text{otherwise.} \end{cases}$$
[These two functions are plotted on the next slide.]
- Since the system is orthonormal, $\tilde{\phi} = \phi$ and $\tilde{\psi} = \psi$.

Haar Scaling and Wavelet Functions



Scaling Function



Wavelet Function

Shannon Wavelet System

- This ***orthonormal*** wavelet system is associated with the Shannon MRA (introduced earlier) which is based on spaces of bandlimited functions.
- The ***scaling function*** ϕ satisfies the refinement equation

$$\phi(t) = \sqrt{2}\sum_{k\in\mathbb{Z}} c_k\phi(2t-k) \quad \text{where} \quad c_k = \begin{cases} \frac{1}{\sqrt{2}} & k=0 \\ \frac{\sqrt{2}(-1)^{(k-1)/2}}{k\pi} & \text{odd } k \\ 0 & \text{even } k,\ k\neq 0. \end{cases}$$

- The ***wavelet function*** ψ can be expressed in terms of ϕ as

$$\psi(t) = \sqrt{2}\sum_{k\in\mathbb{Z}} d_k\phi(2t-k) \quad \text{where} \quad d_k = (-1)^k c_k.$$

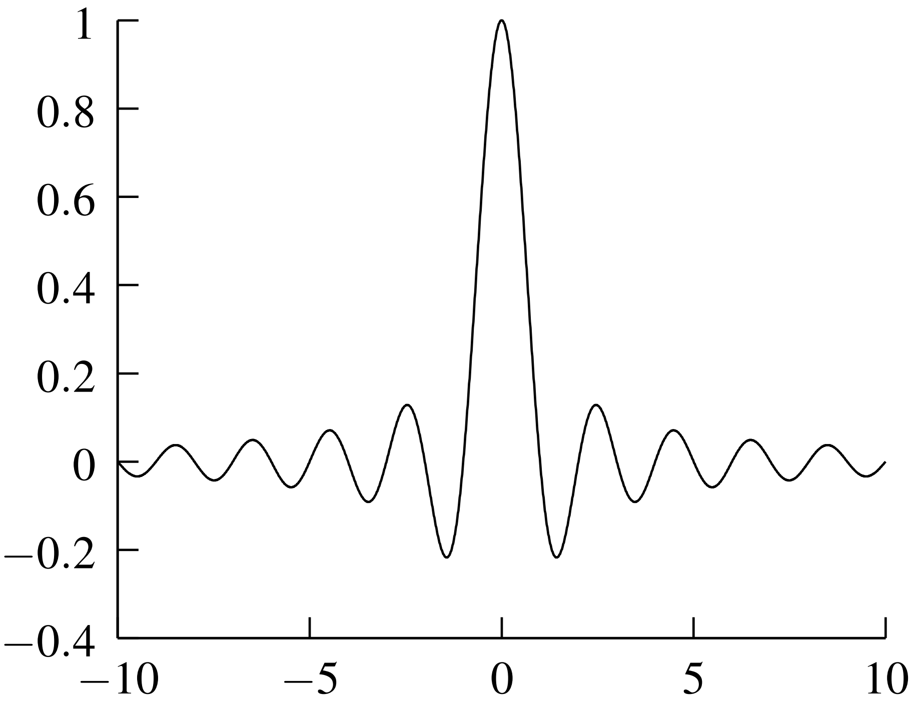
- Fortunately, ϕ and ψ can be expressed in closed form as

$$\phi(t) = \operatorname{sinc}\pi t \quad \text{and} \quad \psi(t) = \left(\cos\tfrac{3\pi}{2}t\right)\operatorname{sinc}\tfrac{\pi}{2}t.$$

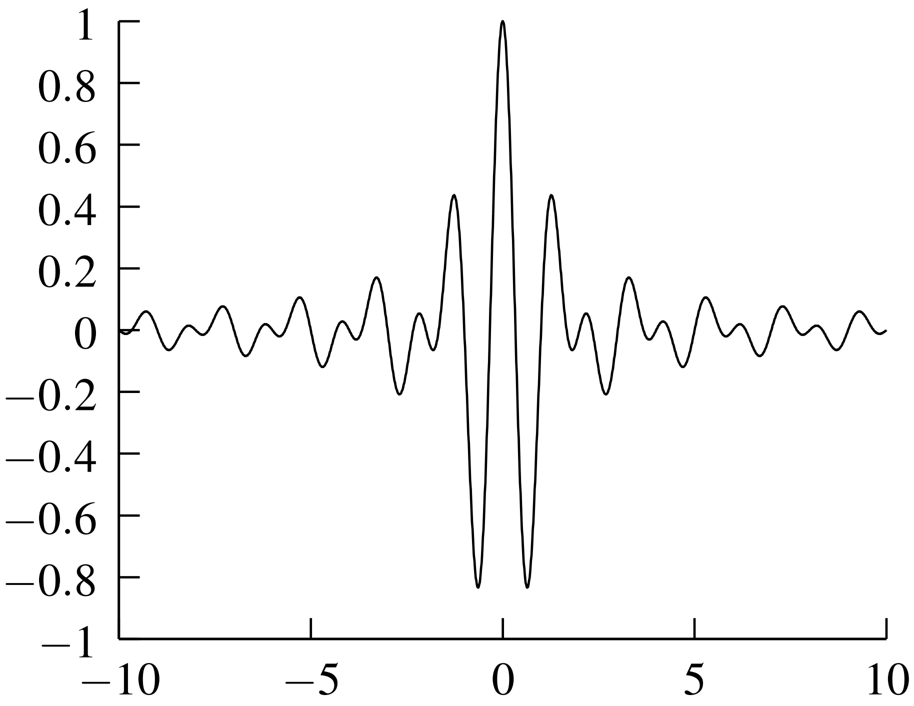
  [These two functions are plotted in on the next slide.]
- Since this system is orthonormal, $\tilde{\phi} = \phi$ and $\tilde{\psi} = \psi$.
- In passing, we note that

$$\hat{\phi}(\omega) = \chi_{[-\pi,\pi]}(\omega) \quad \text{and} \quad \hat{\psi}(\omega) = \chi_{[-2\pi,-\pi)}(\omega) + \chi_{(\pi,2\pi]}(\omega).$$

Shannon Scaling and Wavelet Functions



Scaling Function



Wavelet Function

Daubechies 2 Wavelet System

- One classic example of an ***orthonormal*** wavelet system is the Daubechies 2 wavelet system.
- The ***scaling function*** ϕ satisfies the refinement equation

$$\phi(t) = \sqrt{2} \sum_{k=-1}^{2} c_k \phi(2t-k)$$

where

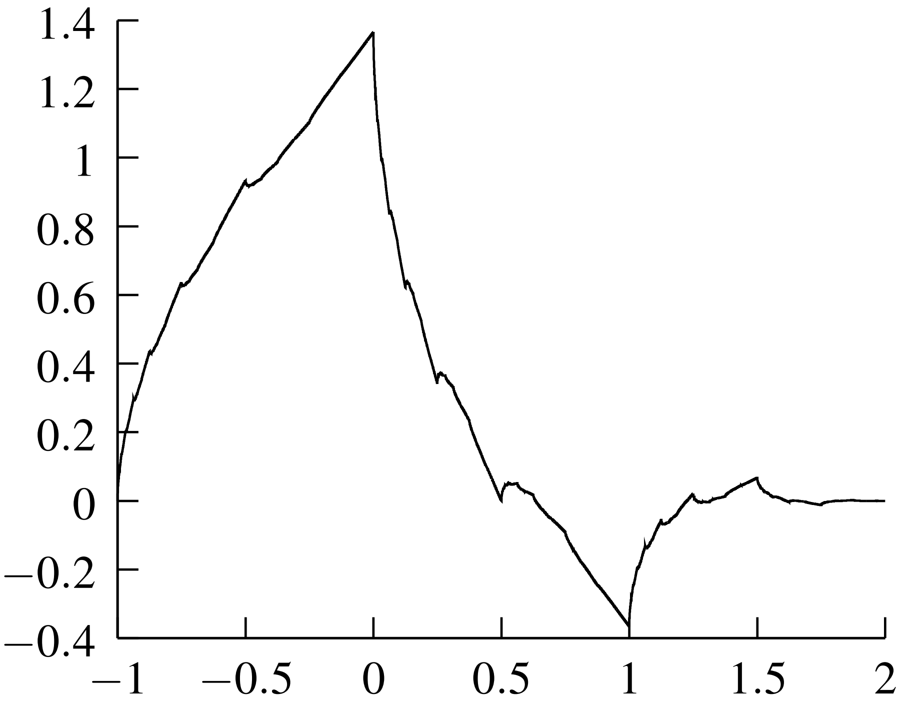
$$c_{-1} = \tfrac{1+\sqrt{3}}{4\sqrt{2}}, \quad c_0 = \tfrac{3+\sqrt{3}}{4\sqrt{2}}, \quad c_1 = \tfrac{3-\sqrt{3}}{4\sqrt{2}}, \quad \text{and} \quad c_2 = \tfrac{1-\sqrt{3}}{4\sqrt{2}}.$$

- The ***wavelet function*** ψ is given by

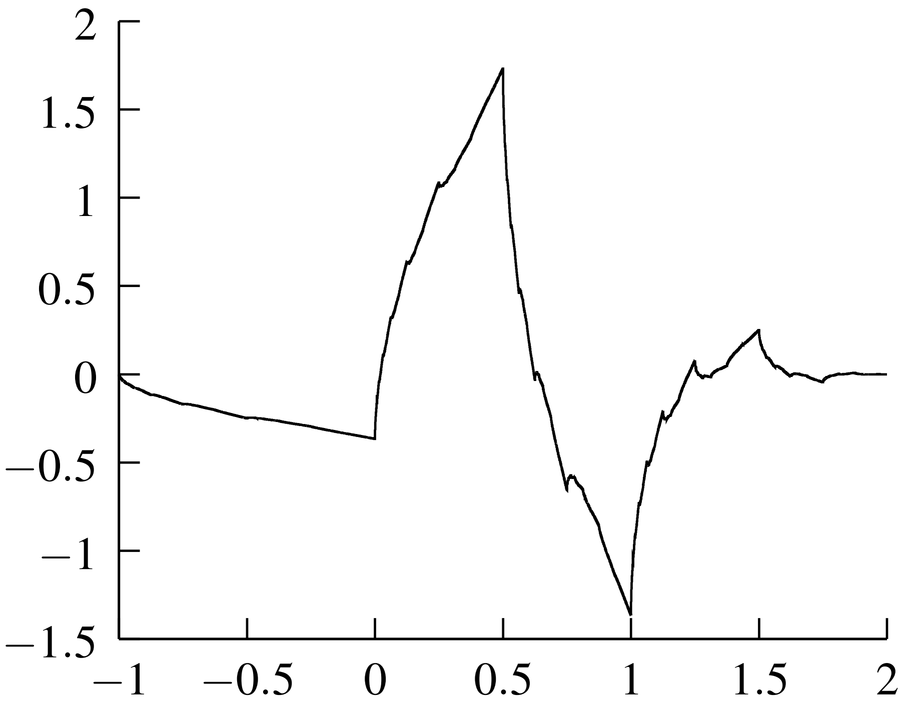
$$\psi(t) = \sqrt{2} \sum_{k=-1}^{2} d_k \phi(2t-k), \quad \text{where} \quad d_k = (-1)^{1-k} c_{1-k}.$$

- Since the system is orthonormal, $\tilde{\phi} = \phi$ and $\tilde{\psi} = \psi$.
- The scaling and wavelet functions are plotted on the next slide.

Daubechies 2 Scaling and Wavelet Functions



Scaling Function



Wavelet Function

Le Gall 5/3 Wavelet System

- This ***biorthonormal*** wavelet system has proven to be extremely useful in signal-coding applications (e.g., image compression).
- This wavelet system is, in part, associated with the linear B-spline MRA. This is the primal MRA.
- The ***primal scaling function*** ϕ satisfies the refinement equation

$$\phi(t) = \sqrt{2} \sum_{k=-1}^{1} c_k \phi(2t-k)$$

where

$$c_{-1} = c_1 = \frac{1}{2\sqrt{2}} \quad \text{and} \quad c_0 = \frac{1}{\sqrt{2}}.$$

- The ***primal wavelet function*** ψ can be expressed in terms of the scaling function as

$$\psi(t) = \sqrt{2} \sum_{k=-1}^{3} d_k \phi(2t-k)$$

where

$$d_{-1} = d_3 = -\frac{1}{4\sqrt{2}}, \quad d_0 = d_2 = -\frac{2}{4\sqrt{2}}, \quad \text{and} \quad d_1 = \frac{6}{4\sqrt{2}}.$$

Le Gall 5/3 Wavelet System (Continued)

- The functions ϕ and ψ can be expressed in closed form as

$$\phi(t) = \begin{cases} 1 - |t| & t \in [-1,1] \\ 0 & \text{otherwise} \end{cases} \quad \text{and}$$

$$\psi(t) = \begin{cases} \frac{3}{2} - 4\left|t - \frac{1}{2}\right| & t \in (0,1) \\ -\frac{3}{4} + \frac{1}{2}\left|t - \frac{1}{2}\right| & t \in (-1,0] \cup [1,2) \\ 0 & \text{otherwise.} \end{cases}$$

- The ***dual scaling function*** $\tilde{\phi}$ satisfies the refinement equation

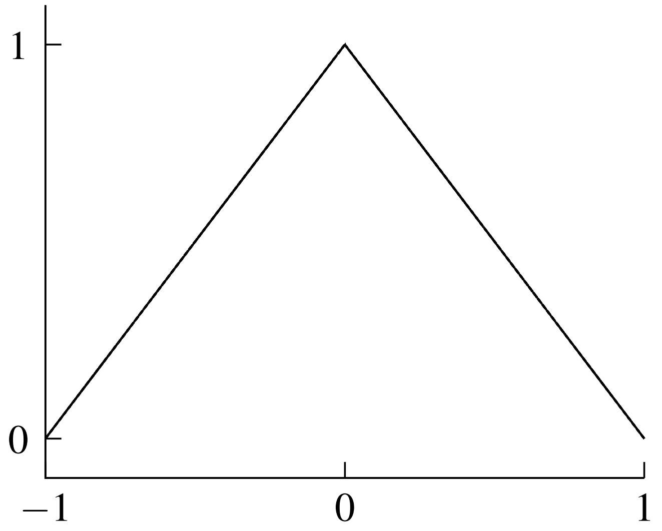
$$\tilde{\phi}(t) = \sqrt{2} \sum_{k=-2}^{2} \tilde{c}_k \tilde{\phi}(2t-k) \quad \text{where} \quad \tilde{c}_k = (-1)^k d_{1-k}.$$

- The ***dual wavelet function*** $\tilde{\psi}$ can be expressed as

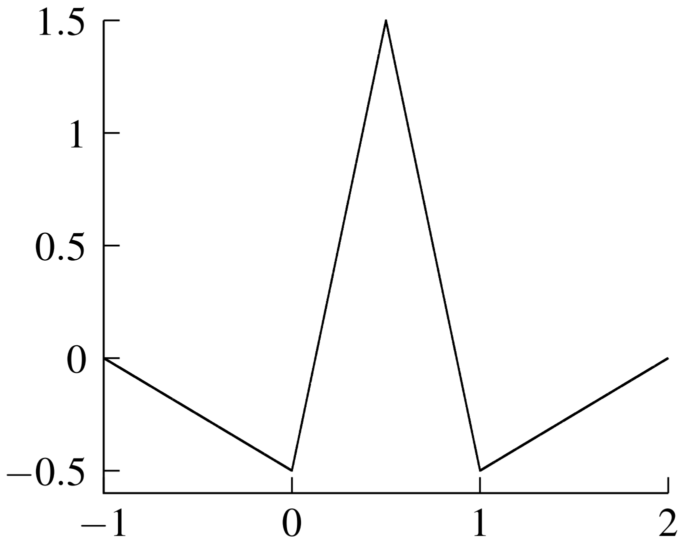
$$\tilde{\psi}(t) = \sqrt{2} \sum_{k=0}^{2} \tilde{d}_k \tilde{\phi}(2t-k) \quad \text{where} \quad \tilde{d}_k = (-1)^{k+1} c_{1-k}.$$

- Unfortunately, there is no closed form expression for $\tilde{\phi}$ and $\tilde{\psi}$.
- Plots of ϕ, ψ, $\tilde{\phi}$, and $\tilde{\psi}$ are shown on the next slide. The plots of the dual functions are somewhat misleading, as one can show that $\tilde{\phi}$ is infinite at every dyadic point.

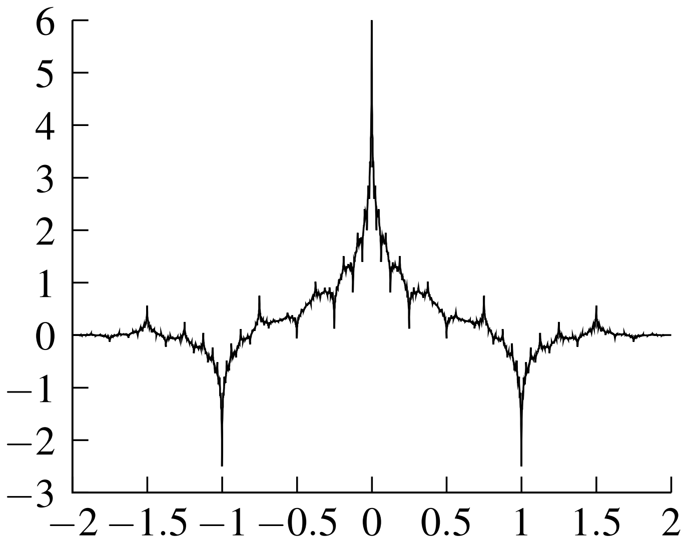
Le Gall 5/3 Scaling and Wavelet Functions



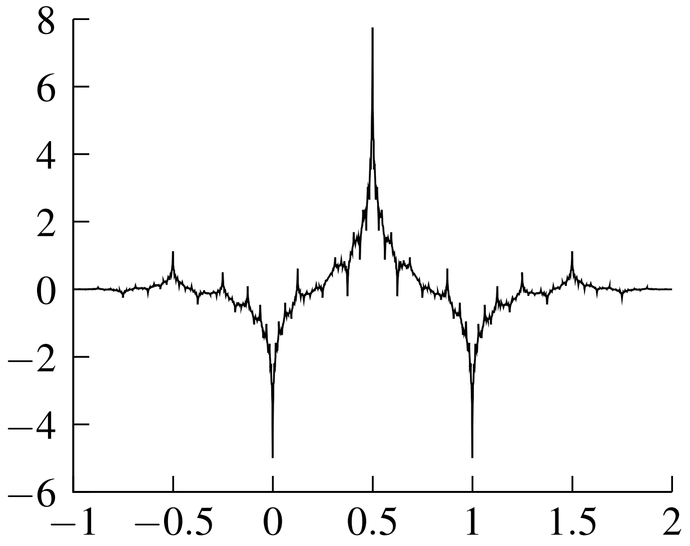
Primal Scaling Function



Primal Wavelet Function



Dual Scaling Function



Dual Wavelet Function

Cohen-Daubechies-Feaveau (CDF) 9/7 Wavelet System

- This ***biorthonormal*** wavelet system has found wide application in signal-coding applications (e.g., JPEG 2000, FBI fingerprint compression standard).
- Define x_1 and x_2 as

$$x_1 = A + B - \tfrac{1}{6} \quad \text{and} \quad x_2 = -\tfrac{1}{2}(A+B) - \tfrac{1}{6} + j\tfrac{\sqrt{3}}{2}(A-B), \quad \text{where}$$

$$A = \sqrt[3]{\tfrac{63-14\sqrt{15}}{1080\sqrt{15}}} \quad \text{and} \quad B = -\sqrt[3]{\tfrac{63+14\sqrt{15}}{1080\sqrt{15}}}.$$

- The ***primal scaling function*** ϕ satisfies the refinement equation

$$\phi(t) = \sqrt{2} \sum_{k=-3}^{3} c_k \phi(2t-k)$$

where

$$c_{-3} = c_3 = \tfrac{1}{32\sqrt{2}\,x_1}, \quad c_{-2} = c_2 = \tfrac{2x_1+1}{16\sqrt{2}\,x_1},$$
$$c_{-1} = c_1 = \tfrac{16x_1-1}{32\sqrt{2}\,x_1}, \quad \text{and} \quad c_0 = \tfrac{6x_1-1}{8\sqrt{2}\,x_1}.$$

CDF 9/7 Wavelet System (Continued)

- The ***primal wavelet function*** ψ is given by

$$\psi(t) = \sqrt{2} \sum_{k=-3}^{5} d_k \phi(2t-k)$$

where

$$\begin{aligned} d_{-3} = d_5 &= -\frac{5}{32\sqrt{2}} x_1, \quad d_{-2} = d_4 = \frac{5}{4\sqrt{2}} x_1 \operatorname{Re} x_2, \\ d_{-1} = d_3 &= -\frac{5}{8\sqrt{2}} x_1 (4\,|x_2|^2 + 4 \operatorname{Re} x_2 - 1), \\ d_0 = d_2 &= \frac{5}{4\sqrt{2}} x_1 (8\,|x_2|^2 - \operatorname{Re} x_2), \quad \text{and} \\ d_1 &= -\frac{5}{16\sqrt{2}} x_1 (48\,|x_2|^2 - 16 \operatorname{Re} x_2 + 3). \end{aligned}$$

- The ***dual scaling function*** $\tilde{\phi}$ satisfies the refinement equation

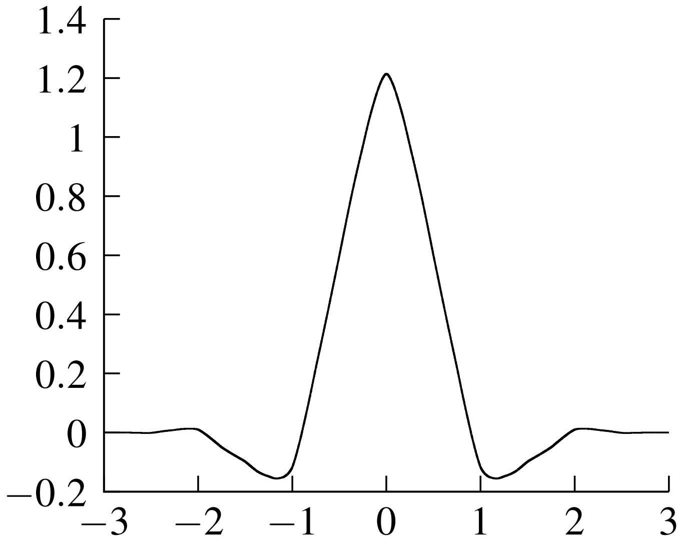
$$\tilde{\phi}(t) = \sqrt{2} \sum_{k=-4}^{4} \tilde{c}_k \tilde{\phi}(2t-k) \quad \text{where} \quad \tilde{c}_k = (-1)^k d_{1-k}.$$

- The ***dual wavelet function*** $\tilde{\psi}$ is given by

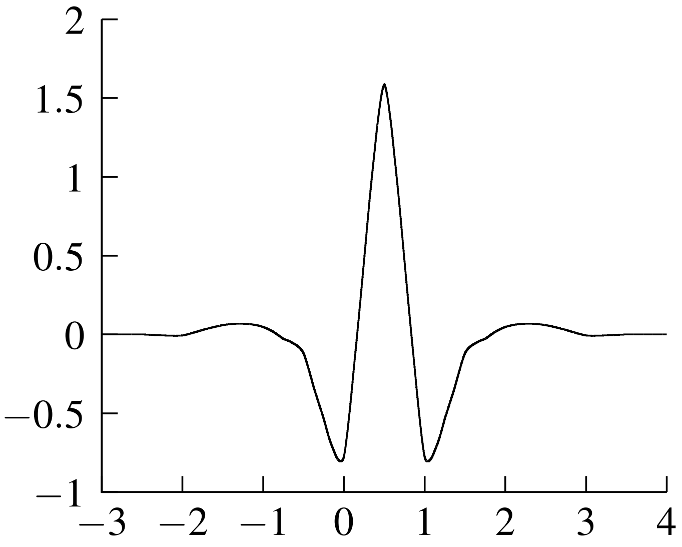
$$\tilde{\psi}(t) = \sqrt{2} \sum_{k=-2}^{4} \tilde{d}_k \tilde{\phi}(2t-k) \quad \text{where} \quad \tilde{d}_k = (-1)^{k+1} c_{1-k}.$$

- The functions ϕ, ψ, $\tilde{\phi}$, and $\tilde{\psi}$ are plotted on the next slide.

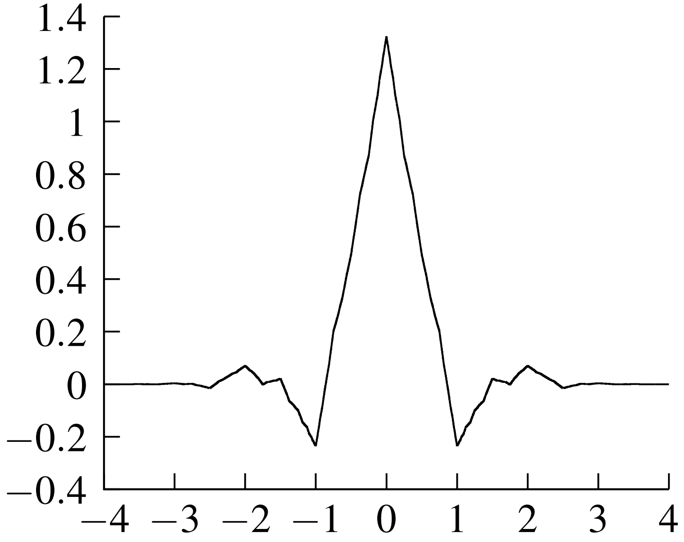
CDF 9/7 Scaling and Wavelet Functions



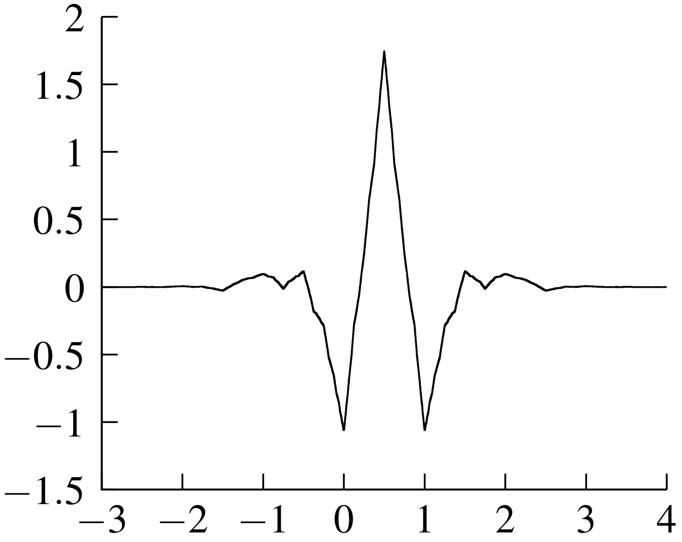
Primal Scaling Function



Primal Wavelet Function



Dual Scaling Function



Dual Wavelet Function

Section 4.7

Wavelets Systems and Filter Banks

A Tale of Two Representations

- Suppose that we have a function $f \in V_p$.
- Since $f \in V_p$, f has an expansion ***in terms of the basis of V_p*** given by

$$f(t) = \sum_{n \in \mathbb{Z}} a_p[n] \phi_{p,n}(t).$$

- Furthermore, as $V_p = V_{p+1} \oplus W_{p+1}$, we can also expand f ***in terms of the bases of V_{p+1} and W_{p+1}*** to obtain

$$f(t) = \sum_{n \in \mathbb{Z}} a_{p+1}[n] \phi_{p+1,n}(t) + \sum_{n \in \mathbb{Z}} b_{p+1}[n] \psi_{p+1,n}(t).$$

- Thus, we have ***two different representations*** of f.
- One might wonder if there exists a simple technique for ***converting*** between these representations.
- In other words, we would like to be able to:
  1. determine a_{p+1} and b_{p+1}, given a_p; or
  2. determine a_p, given a_{p+1} and b_{p+1}.
- Fortunately, there is a very elegant technique for accomplishing exactly this. This technique is known as the ***Mallat algorithm***.

Mallat Algorithm

- **Theorem.** Consider a wavelet system with approximation space sequence $\{V_p\}_{p\in\mathbb{Z}}$, scaling function ϕ, wavelet space sequence $\{W_p\}_{p\in\mathbb{Z}}$, wavelet function ψ, dual scaling function $\tilde{\phi}$, and dual wavelet function $\tilde{\psi}$. Let the scaling equation coefficient sequences of ϕ and $\tilde{\phi}$, and wavelet equation coefficient sequences of ψ and $\tilde{\psi}$ be denoted as c, $\tilde{c}$, d, and $\tilde{d}$, respectively. Let $\{\phi_{p,n}\}_{n\in\mathbb{Z}}$ and $\{\psi_{p,n}\}_{n\in\mathbb{Z}}$ denote the bases of V_p and W_p, respectively, where $\phi_{p,n}(t) = 2^{-p/2}\phi(2^{-p}t-n)$ and $\psi_{p,n}(t) = 2^{-p/2}\psi(2^{-p}t-n)$. Any $f \in V_p$ can be represented in each of the following forms:

$$f = \sum_{n\in\mathbb{Z}} a_p[n]\phi_{p,n} \quad \text{and} \quad f = \sum_{n\in\mathbb{Z}} a_{p+1}[n]\phi_{p+1,n} + \sum_{n\in\mathbb{Z}} b_{p+1}[n]\psi_{p+1,n}.$$

Given a_p, we can compute the corresponding a_{p+1} and b_{p+1} as

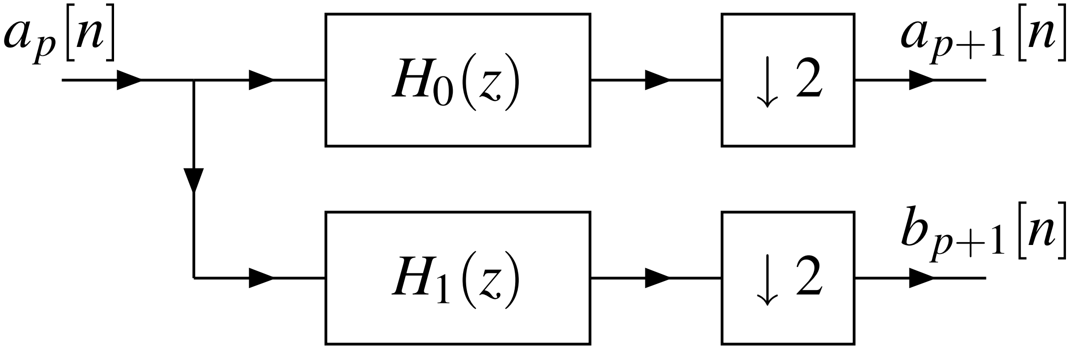
$$a_{p+1} = (\downarrow 2)\left(a_p * h_0\right) \quad \text{and} \quad b_{p+1} = (\downarrow 2)\left(a_p * h_1\right), \quad \text{where}$$
$$h_0[n] = \tilde{c}^*[-n] \quad \text{and} \quad h_1[n] = \tilde{d}^*[-n].$$

Given a_{p+1} and b_{p+1}, we can compute the corresponding a_p as

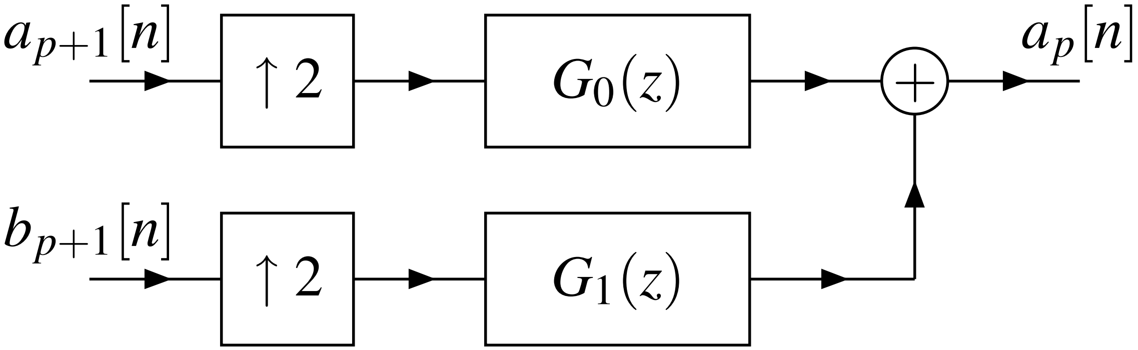
$$a_p = \left[(\uparrow 2)a_{p+1}\right] * g_0 + \left[(\uparrow 2)b_{p+1}\right] * g_1, \quad \text{where}$$
$$g_0[n] = c[n] \quad \text{and} \quad g_1[n] = d[n].$$

Computational Structures for Mallat Algorithm

- The Mallat algorithm is associated with a ***two-channel UMD filter bank***.
- The Mallat algorithm is of great importance as it establishes a ***link between wavelet systems and filter banks***.



Compute a_{p+1} and b_{p+1} from a_p



Compute a_p from a_{p+1} and b_{p+1}

Characterization of Biorthonormal Wavelet Systems

- Let $\{\phi_{p,k}\}_{p\in\mathbb{Z}}$, $\{\tilde{\phi}_{p,k}\}_{p\in\mathbb{Z}}$, $\{\psi_{p,k}\}_{p\in\mathbb{Z}}$, and $\{\tilde{\psi}_{p,k}\}_{p\in\mathbb{Z}}$ denote the Riesz bases of V_p, $\tilde{V}_p$, W_p, $\tilde{W}_p$, respectively, where $\phi_{p,k}$, $\tilde{\phi}_{p,k}$, $\psi_{p,k}$, $\tilde{\psi}_{p,k}$ are as defined earlier.
- A biorthonormal system is such that

$$\left\langle \phi_{p,k}, \tilde{\phi}_{p,l} \right\rangle = \delta[k-l], \quad \langle \psi_{p,k}, \tilde{\psi}_{p,l} \rangle = \delta[k-l],$$
$$\langle \phi_{p,k}, \tilde{\psi}_{p,l} \rangle = 0, \quad \text{and} \quad \left\langle \psi_{p,k}, \tilde{\phi}_{p,l} \right\rangle = 0.$$

- One can show that biorthonormality is equivalent to the following condition expressed in terms of c, d, $\tilde{c}$, and $\tilde{d}$:

$$\langle c[\cdot], \tilde{c}[\cdot-2n] \rangle = \delta[n], \quad \left\langle c[\cdot], \tilde{d}[\cdot-2n] \right\rangle = 0,$$
$$\left\langle d[\cdot], \tilde{d}[\cdot-2n] \right\rangle = \delta[n], \quad \text{and} \quad \langle d[\cdot], \tilde{c}[\cdot-2n] \rangle = 0.$$

Characterization of Biorthonormal Wavelet Systems (Continued)

- Since $g_0[n] = c[n]$, $g_1[n] = d[n]$, $h_0[n] = \tilde{c}^*[-n]$, and $\tilde{d}^*[-n]$, we can rewrite the conditions from the previous slide as:

$$\langle g_0[\cdot], h_0^*[2n-\cdot]\rangle = \delta[n], \quad \langle g_0[\cdot], h_1^*[2n-\cdot]\rangle = 0,$$
$$\langle g_1[\cdot], h_1^*[2n-\cdot]\rangle = \delta[n], \quad \text{and} \quad \langle g_1[\cdot], h_0^*[2n-\cdot]\rangle = 0,$$

  or equivalently,

$$\langle g_k[\cdot], h_l^*[2n-\cdot]\rangle = \delta[k-l]\delta[n].$$

- As it turns out, the above condition is a restatement of the ***shift-free PR*** condition for a UMD filter bank.
- Therefore, a biorthogonal wavelet system is associated with a ***shift-free PR filter bank*** (via the Mallat algorithm).

Characterization of Orthonormal Wavelet Systems

- If the wavelet system is orthonormal, then $\tilde{c} = c$ and $\tilde{d} = d$.
- Thus, orthonormality is equivalent to the following conditions on c and d:

$$\begin{gathered}\langle c[\cdot], c[\cdot - 2n]\rangle = \delta[n], \quad \langle c[\cdot], d[\cdot - 2n]\rangle = 0, \\ \langle d[\cdot], d[\cdot - 2n]\rangle = \delta[n], \quad c = \tilde{c}, \quad \text{and} \quad d = \tilde{d}.\end{gathered}$$

- Expressed in terms of impulse responses, the above condition becomes:

$$\langle g_k[\cdot], g_l[\cdot - 2n]\rangle = \delta[k-l]\delta[n], \quad h_k[n] = g_k^*[-n].$$

- As it turns out, the above condition is a restatement of the ***orthonormal shift-free PR*** condition for a UMD filter bank.
- Therefore, an orthonormal wavelet system is associated with an ***orthonormal shift-free PR filter bank*** (via the Mallat algorithm).

Importance of Basis Interpretation

- Suppose that we have a function f represented in terms of the basis functions $\{\varphi_n\}_{n\in\mathbb{Z}}$ as

$$f(t) = \sum_{n\in\mathbb{Z}} a_n \varphi_n(t)$$

- In practice, we can only compute a finite sum, so we must drop terms in the above summation, effectively changing some of the $\{a_n\}_{n\in\mathbb{Z}}$ from a nonzero value to zero.
- Also, in practice, we often do not represent the $\{a_n\}_{n\in\mathbb{Z}}$ exactly.
- When we change a coefficient a_n, what error does this introduce?
- If we change a_k to a'_k, the kth term changes from $a_k\varphi_k$ to $a'_k\varphi_k$ (i.e., the error is $(a'_k - a_k)\varphi_k$).
- The error has the same shape as the basis function.

Section 4.8

Properties of Scaling and Wavelet Functions

Properties of Scaling Function

- **Theorem.** Suppose that ϕ is a compactly supported function with zeroth moment $\mu_0 \neq 0$ and $\{\phi(\cdot - k)\}_{k \in \mathbb{Z}}$ is a Riesz basis, and ϕ satisfies a refinement equation with mask $\sqrt{2}c[n]$. Then, we have

$$\begin{aligned} \frac{1}{\mu_0} \sum_{k \in \mathbb{Z}} \phi(t-k) = 1 \;\; (\textbf{partition of unity}), \\ \hat{\phi}(2\pi k) = 0 \;\; \text{for all } k \in \mathbb{Z} \setminus \{0\}, \\ \hat{c}(0) = \sqrt{2}, \quad \text{and} \\ \hat{c}(\pi) = 0. \end{aligned}$$

- The partition of unity property is quite remarkable.
- Regardless of the complexity of the appearance of a scaling function ϕ, the integer translates of ϕ always sum to a constant (namely, μ_0).
- Consequently, integer translates of the scaling function can be used to (locally) reproduce constant functions.

Zeroth Moment of Primal and Dual Scaling Functions

- **Theorem.** If $\phi, \tilde{\phi} \in L^1(\mathbb{R}) \cap L^2(\mathbb{R})$ are biorthogonal scaling functions, then
$$\mu_0 = 1/(\tilde{\mu}_0^*),$$
where μ_0 and $\tilde{\mu}_0$ are the zeroth moments of ϕ and $\tilde{\phi}$, respectively.
- Since a scalar multiple of a solution to a refinement equation is also a solution, there is some freedom in how we choose to normalize the scaling function of a MRA (i.e., how we select the zeroth moment of the function).
- The above theorem is significant because it shows what normalization of a dual scaling function is associated with a particular normalization of a primal scaling function.

Zeroth Moment of Wavelet Function

- **Theorem.** Let ψ be a wavelet function with the wavelet equation coefficient sequence d. Then,

$$\int_{\mathbb{R}} \psi(t)dt = 0$$

(i.e., the zeroth moment of ψ is zero) and

$$\hat{d}(0) = 0.$$

Relationship Between Continuous and Discrete Moments

- **Theorem.** Let ϕ be a scaling function with scaling equation coefficient sequence c. Let ψ be a corresponding wavelet function with wavelet equation coefficient sequence d. Denote the kth moments of ϕ and ψ as μ_k and ν_k, respectively. Denote the kth moments of c and d as m_k and n_k, respectively. Then, we have

$$\mu_k = \frac{1}{2^{1/2}(2^k-1)} \sum_{q=0}^{k-1} \binom{k}{q} m_{k-q}\mu_q \quad \text{for } k \geq 1 \quad \text{and}$$

$$\nu_k = 2^{-k-1/2} \sum_{q=0}^{k} \binom{k}{q} n_{k-q}\mu_q \quad \text{for } k \geq 0.$$

- The above result is quite significant from a practical perspective.
- Usually, ϕ and ψ cannot be expressed in closed form. In spite of this, however, we would often like to know the moments of these functions.
- The above theorem provides a means to calculate the moments of ϕ and ψ from their corresponding coefficient sequences c and d, without ever needing to explicitly compute ϕ or ψ.

Equivalence of Vanishing Continuous and Discrete Moments

- Often, we are interested in the specific case when certain moments of the wavelet and/or scaling function are zero. The below theorem is useful in this regard.
- **Theorem.** Let ϕ be a scaling function with scaling equation coefficient sequence c. Let ψ be the corresponding wavelet function with wavelet equation coefficient sequence d. Denote the kth moments of ϕ and ψ as μ_k and ν_k, respectively. Denote the kth moments of c and d as m_k and n_k, respectively. Then, we have that

$$m_k = 0 \text{ for } k = 1, 2, \ldots, \eta - 1 \Leftrightarrow \mu_k = 0 \text{ for } k = 1, 2, \ldots, \eta - 1; \quad \text{and}$$
$$n_k = 0 \text{ for } k = 0, 1, \ldots, \eta - 1 \Leftrightarrow \nu_k = 0 \text{ for } k = 0, 1, \ldots, \eta - 1$$

(i.e., the first η moments of ϕ vanishing excluding the zeroth moment is equivalent to the first η moments of c vanishing excluding the zeroth moment; and the first η moments of ψ vanishing is equivalent to the first η moments of d vanishing).

Support of Scaling and Wavelet Functions

- As it turns out, a relationship exists between the support of a scaling/wavelet function and the support of its corresponding scaling/wavelet equation coefficient sequence.
- **Theorem.** Let ϕ be a scaling function with scaling equation coefficient sequence c. Let ψ be the corresponding wavelet function with wavelet equation coefficient sequence d. If $c[n] = 0$ whenever $n < N_1$ or $n > N_2$, then

$$\operatorname{supp}\phi \subset [N_1, N_2].$$

If, in addition, $d[n] = 0$ for $n < M_1$ and $n > M_2$, then

$$\operatorname{supp}\psi \subset \left[\frac{N_1+M_1}{2}, \frac{N_2+M_2}{2}\right].$$

- **Theorem.** Let ϕ be a scaling function with scaling equation coefficient sequence c, where $c[n] = 0$ whenever $c < N_1$ or $c > N_2$. If $c[N_1] \neq \frac{1}{\sqrt{2}}$, then $\phi(N_1) = 0$. If $c[N_2] \neq \frac{1}{\sqrt{2}}$, then $\phi(N_2) = 0$.

Wavelets Systems and Sparse Representations

- Polynomials are a useful tool for the approximation of many classes of functions.
- Since polynomials are often a useful tool for approximation, one might be interested in knowing how well polynomials can be approximated by the bases associated with a wavelet system.
- As we have already seen, the integer translates of a scaling function can reproduce constant functions.
- In some cases, however, it so happens that scaling functions can exactly represent polynomials of higher orders.
- Suppose, for a moment, that the integer translates of a scaling function ϕ could reproduce polynomials of order η.
- Consider the representation of a function f that is well approximated by a polynomial of order η, such as a piecewise smooth function.
- Since ϕ can exactly reproduce polynomials of order η, the wavelet coefficients will essentially be zero over all regions where f is well approximated by a polynomial of order η.
- In this way, very sparse representations can be obtained for piecewise smooth functions (i.e., representations with relatively few nonzero expansion coefficients).
- This has major advantages in a broad spectrum of applications.

Order of Approximation

- The polynomial approximation properties of a scaling function are characterized by the theorem below.
- **Theorem.** Let ϕ be a scaling function with the scaling equation coefficient sequence c. Let $\tilde{\psi}$ denote the corresponding dual wavelet function with the wavelet equation coefficient sequence $\tilde{d}$. Suppose that $\hat{\phi}$ and $\hat{\tilde{\psi}}$ are $\eta - 1$ times differentiable. Then, the following statements are equivalent:
  1. ϕ has approximation order η (i.e., $\{\phi(\cdot - n)\}_{n \in \mathbb{Z}}$ can represent kth degree polynomials exactly for $k < \eta$);
  2. for any $0 \leq k < \eta$, $q_k(t) = \sum_{n \in \mathbb{Z}} n^k \phi(t - n)$ is a kth degree polynomial;
  3. $\hat{c}(\omega)$ has a ηth order zero at $\omega = \pi$;
  4. $\tilde{\psi}$ has η vanishing moments (i.e., $\tilde{\psi}$ has all of its moments of order less than η vanish);
  5. $\hat{\tilde{d}}(\omega)$ has a ηth order zero at $\omega = 0$.

Order of Approximation (Continued)

- We can derive an expression for t^p in terms of ϕ for $p < \eta$.
- For example, one can show that

$$t^p = \sum_{n \in \mathbb{Z}} a_p[n] \phi(t-n),$$

where

$$a_p[n] = \begin{cases} \frac{1}{\mu_0} & \text{for } p = 0 \\ \frac{1}{\mu_0} n + \frac{\mu_1}{\mu_0^2} & \text{for } p = 1 \\ \frac{1}{\mu_0} n^2 + \frac{2\mu_1}{\mu_0^2} n + \frac{-\mu_2\mu_0 + 2\mu_1^2}{\mu_0^3} & \text{for } p = 2. \end{cases}$$

- The above formula is only valid if ϕ has approximation order η and $p < \eta$.

Sum Rule

- Since we are clearly interested in the number of zeros that $\hat{c}(\omega)$ has at $\omega = \pi$, it is convenient to have a method to quickly determine this quantity.
- To this end, we will often find the result below to be helpful.
- **Theorem.** Let c be a sequence. Then, $\hat{c}(\omega)$ has a ηth order zero at $\omega = \pi$ if and only if

$$\sum_{n \in \mathbb{Z}} (-1)^n n^k c[n] = 0 \text{ for } k = 0, 1, \ldots, \eta - 1.$$

- This is often referred to as the **sum rule** of order η.

Section 4.9

Determination of Scaling and Wavelet Functions

Determination of Scaling and Wavelet Functions

- Often, it is not possible to find a closed-form expression for scaling and wavelet functions.
- In such cases, we must resort to numerical techniques in order to find these functions (or approximations thereof).
- Since the wavelet function can be expressed in terms of the scaling function, finding the wavelet function degenerates into a problem of finding the scaling function.
- To find a scaling function, we need to solve a refinement equation.
- Thus, we need techniques for solving refinement equations.
- In what follows, we consider methods for numerically determining solutions to refinement equations.

Spectral Method

- Conceptually, one of the simplest methods for computing the scaling function is to use the infinite product formula for the Fourier transform of the scaling function.
- Recall that we can write

$$\hat{\phi}(\omega) = \hat{\phi}(0) \prod_{p=1}^{\infty} \tfrac{1}{\sqrt{2}} \hat{c}(\omega/2^p).$$

- So, in principle, we can approximate the above infinite product by its first k factors.
- Then, we can take the inverse Fourier transform of the result in order to find an approximation to $\phi(t)$.
- While this approach works, it is ***neither very exact nor particularly fast***.
- This leads us to consider other methods for determining the scaling function.

Cascade Algorithm

- Another approach to determining the scaling function involves repeated application of a filter bank and is known as the cascade algorithm.
- This algorithm is fast and, although not exact, often yields a good approximation.
- This approach is formalized by the theorem below.
- **Theorem.** Suppose that we have a refinement equation

  $$\phi(t) = \sqrt{2} \sum_{n \in \mathbb{Z}} c[n] \phi(2t - n).$$

  Define the iterative process

  $$\phi^{(k+1)}(t) = \sqrt{2} \sum_{n \in \mathbb{Z}} c[n] \phi^{(k)}(2t - n)$$

  where we choose $\phi^{(0)}(t)$ such that

  $$\int_{-\infty}^{\infty} \phi^{(0)}(t) dt = \mu_0 \neq 0.$$

  If this iterative process converges to a fixed point, this fixed point is a solution to the above refinement equation normalized such that $\int_{-\infty}^{\infty} \phi(t) dt = \mu_0$.

Cascade Algorithm (Continued)

- In practice, we usually choose $\phi^{(0)} = \chi_{[0,1)}$.
- With such a choice, we then have that $\phi^{(k)}$ is of the form

$$\phi^{(k)}(t) = \sum_{n \in \mathbb{Z}} a^{(k)}[n] \chi_{[n2^{-k},(n+1)2^{-k})}(t).$$

  That is, $\phi^{(k)}$ is piecewise constant on intervals of the form $[n2^{-k},(n+1)2^{-k})$, where $n \in \mathbb{Z}$.
- Furthermore, one can show that the sequence $a^{(k)}$ is given by

$$a^{(k)}[n] = \begin{cases} \left(\left[(\uparrow 2)a^{(k-1)} \right] * (\sqrt{2}c) \right)[n] & \text{for } k \geq 1 \\ \delta[n] & \text{for } k = 0. \end{cases}$$

- The above algorithm can be implemented very conveniently in software. We need only compute the sequence $a^{(\kappa)}$ for some sufficiently large κ.
- From the resulting sequence, we can then trivially deduce $\phi^{(\kappa)}$.

Eigenmethod

- Another technique for solving refinement equations determines the solution at dyadic points.
- First, we solve an eigenproblem to find the solution at integers.
- Then, we employ the scaling equation to compute the solution at half integers, quarter integers, and so on.
- This approach is both exact and fast. It can be used to evaluate the solution at any dyadic point.

Part 5

Geometry Processing Preliminaries

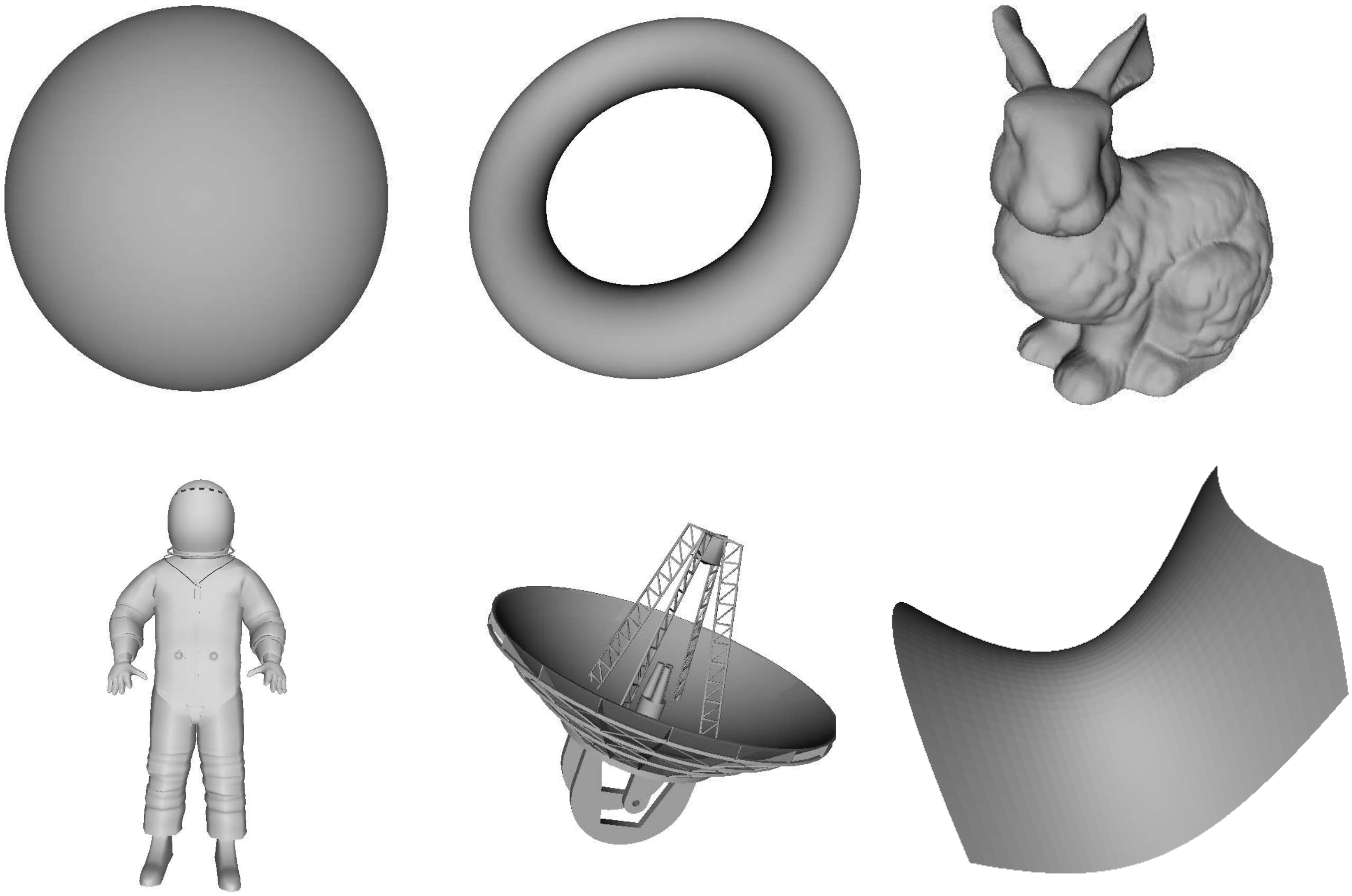
Digital Geometry Processing

- As digital computing devices have become more powerful, the complexity of the datasets processed with these devices has also increased.
- First came digital audio, then digital imagery, and then digital video.
- More recently, we have seen the rise of digital geometry, that is, digital representations of geometric objects such as polyhedra and surfaces.
- Digital geometry processing deals with the ***representation and manipulation*** of geometric objects in digital form.
- Digital geometry processing has a ***wide range of application areas***, including:
  - multimedia, animation, gaming
  - biomedical computing
  - scientific visualization
  - geometric modelling
  - computer-aided design and manufacturing
  - finite element analysis
  - computational fluid dynamics

Geometry Processing Versus Traditional Signal Processing

- In the case of traditional signal processing, signals are essentially ***functions*** defined on a Euclidean domain $\mathbb{R}^n$.
- For example, an audio signal is a function defined on $\mathbb{R}$, where the domain of the function corresponds to time.
- An image signal is a function defined on $\mathbb{R}^2$, where the domain of the function corresponds to horizontal and vertical position.
- A video signal can be viewed as a function defined on $\mathbb{R}^3$, where the domain of the function corresponds to horizontal position, vertical position, and time.
- In the case of geometry processing, the mathematical objects of interest are not functions. Rather, these objects are typically what are known as ***manifolds*** (with or without boundaries).

Examples of Geometric Objects



Section 5.1

Linear Algebra, Affine Geometry, and Projective Geometry

Cross Product

- The **cross product** of two vectors $v = (v_1, v_2, v_3)$ and $w = (w_1, w_2, w_3)$ in $\mathbb{R}^3$, denoted $v \times w$, is defined as

  $$\det \begin{bmatrix} i & j & k \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{bmatrix},$$

  where i, j, and k denote unit vectors in the x, y, and z directions, respectively.
- **Example.** Let $v = (1,2,3)$ and $w = (1,-2,1)$. Then,

  $$v \times w = (1,2,3) \times (1,-2,1) = \det \begin{bmatrix} i & j & k \\ 1 & 2 & 3 \\ 1 & -2 & 1 \end{bmatrix} =$$

  $$i(2+6) - j(1-3) + k(-2-2) = 8i + 2j - 4k = (8,2,-4).$$
- For all $v, w \in \mathbb{R}^3$, $v \times w \perp v$ and $v \times w \perp w$.
- The cross product provides a means to find a vector perpendicular to two other vectors.
- If v and w are noncollinear vectors parallel to a plane P, a ***normal*** (vector) n to P is given by $n = v \times w$.

Affine Combinations and Affine Hull

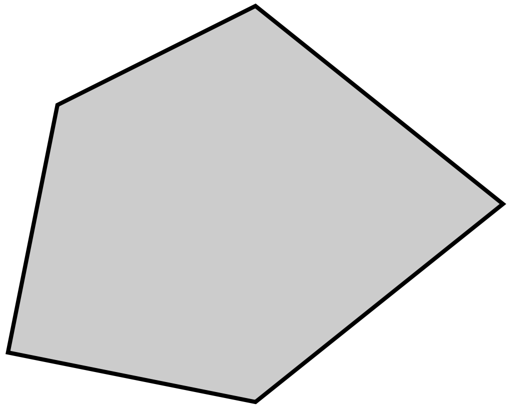
- An **affine combination** of vectors $v_1, v_2, \ldots, v_n$ in a vector space V over the field F is an expression of the form

$$\sum_{k=1}^{n} a_k v_k \quad \text{where} \quad \sum_{k=1}^{n} a_k = 1,$$

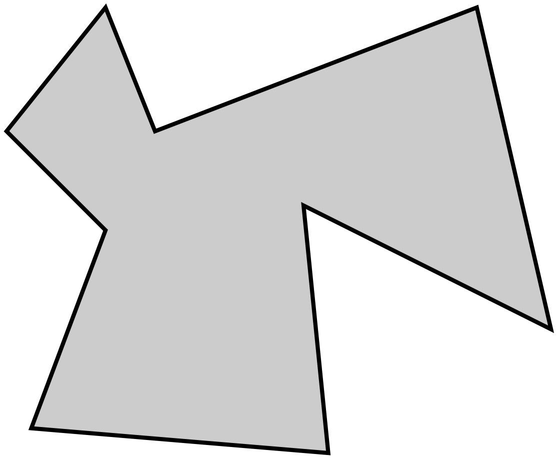
  and $a_1, a_2, \ldots, a_n \in F$ (i.e., an affine combination is a linear combination for which the sum of the coefficients is one).
- The **affine hull** of $X \subset \mathbb{R}^n$, denoted $\operatorname{aff} X$, is the intersection of all hyperplanes in $\mathbb{R}^n$ that contain X.
- Equivalently, the affine hull of $X \subset \mathbb{R}^n$ is the set of ***all affine combinations*** of elements in X.
- **Example.** The affine hull of a set of two distinct points is the line through them.
- **Example.** The affine hull of a set of three non-collinear points is the plane through them.
- **Example.** The affine hull of a set of four non-coplanar points is $\mathbb{R}^3$.

Convex Set

- A set S of points (in $\mathbb{R}^2$) is **convex** if, for any two points $p, q \in S$, the line segment with endpoints p and q is contained in S.
- Examples of convex and nonconvex sets are shown below.



Convex Set



Nonconvex Set

Convex Combinations and Convex Hull

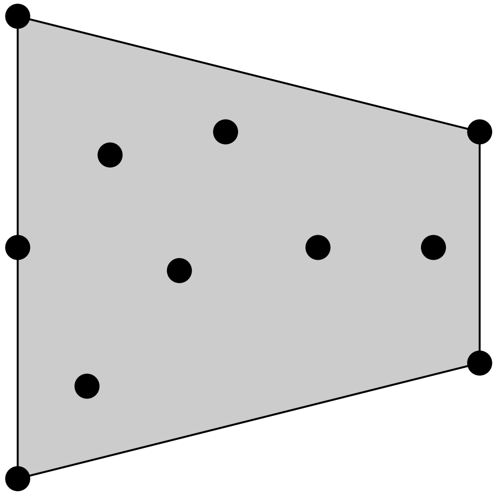
- A **convex combination** of vectors $v_1, v_2, \ldots, v_n$ in a real vector space V is an expression of the form

  $$\sum_{k=1}^{n} a_k v_k \quad \text{where} \quad \sum_{k=1}^{n} a_k = 1 \quad \text{and} \quad a_k \geq 0$$

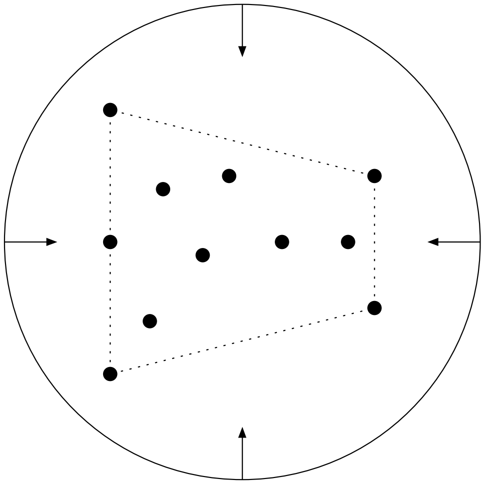
  (i.e., a convex combination is simply an affine combination with nonnegative coefficients). (Note that $\sum_{k=1}^{n} a_k = 1$ and $a_k \geq 0$ together imply that $a_k \in [0, 1]$.)
- The **convex hull** of $X \subset \mathbb{R}^n$, denoted $\operatorname{conv} X$, is defined as the intersection of all convex sets containing X (i.e., the smallest convex set that contains X).
- Equivalently, the convex hull of $X \subset \mathbb{R}^n$ is the set of ***all convex combinations*** of elements in X.
- **Example.** Consider a triangle T whose vertices v_1, v_2, v_3 lie in the plane P in $\mathbb{R}^3$. The convex hull of T is the interior of the triangle plus its boundary, while the affine hull of T is the entire plane P.

Convex Hull (Continued)

- The boundary of H is a ***convex polygon*** with vertices in P.
- The boundary of H can be visualized in terms of ***elastic band/sheet*** stretched to encompass the points in P.
- A point $p \in H$ that does not lie on any open line segment joining two points in P is called an **extreme point** (i.e., "corner").



Convex Hull



Elastic band visualization of convex-hull boundary

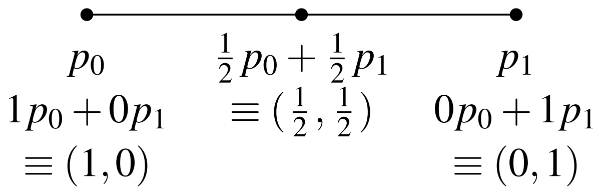
Barycentric Coordinates

- **Theorem.** Let $v_0, v_1, \ldots, v_m$ be $m+1$ linearly independent points in $\mathbb{R}^n$ (where, clearly, $m \leq n$). Every point in the convex hull of $v_0, v_1, \ldots, v_m$ can be expressed ***uniquely*** as a convex combination of $v_0, v_1, \ldots, v_m$. That is, every point w in the convex hull of $v_0, v_1, \ldots, v_m$ has a unique representation of the form

$$w = \sum_{k=0}^{m} a_k v_k,$$

  where $a_k \in [0,1]$ and $\sum_{k=0}^{m} a_k = 1$.
- The $\{a_k\}_{k \in \{1,2,\ldots,n\}}$ above are called the **barycentric coordinates** of w with respect to the points $v_1, v_2, \ldots, v_m$.

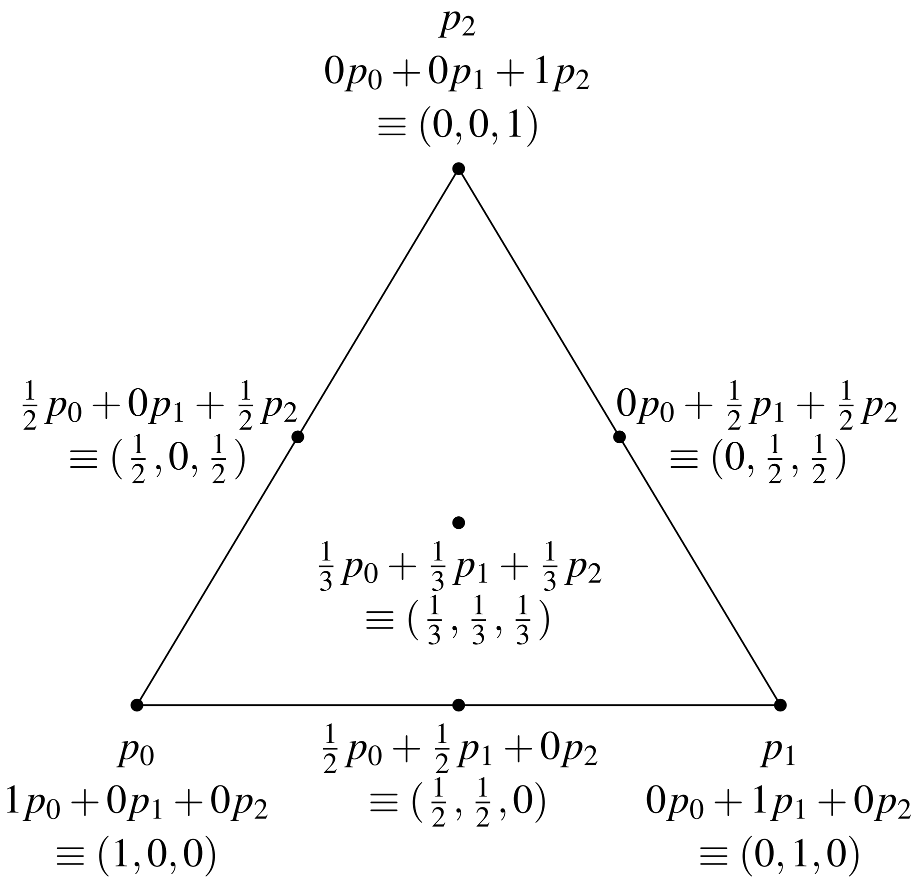
Barycentric Coordinates Example



p_0
$1p_0 + 0p_1$
$\equiv (1,0)$

$\frac{1}{2}p_0 + \frac{1}{2}p_1$
$\equiv (\frac{1}{2}, \frac{1}{2})$

p_1
$0p_0 + 1p_1$
$\equiv (0,1)$



p_2
$0p_0 + 0p_1 + 1p_2$
$\equiv (0,0,1)$

$\frac{1}{2}p_0 + 0p_1 + \frac{1}{2}p_2$
$\equiv (\frac{1}{2}, 0, \frac{1}{2})$

$0p_0 + \frac{1}{2}p_1 + \frac{1}{2}p_2$
$\equiv (0, \frac{1}{2}, \frac{1}{2})$

$\frac{1}{3}p_0 + \frac{1}{3}p_1 + \frac{1}{3}p_2$
$\equiv (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$

p_0
$1p_0 + 0p_1 + 0p_2$
$\equiv (1,0,0)$

$\frac{1}{2}p_0 + \frac{1}{2}p_1 + 0p_2$
$\equiv (\frac{1}{2}, \frac{1}{2}, 0)$

p_1
$0p_0 + 1p_1 + 0p_2$
$\equiv (0,1,0)$

Affine Transformations

- A one-to-one and onto mapping $T : \mathbb{R}^n \to \mathbb{R}^n$ that preserves the collinearity of points (i.e., maps lines onto lines) is called an **affine transformation**.
- Affine transformations include scalings, rotations, shears, translations, and compositions thereof.
- Every (invertible) linear transformation is an affine transformation.
- Affine transformations ***preserve convex sets***.
- For any affine transformation T and any finite set $\{p_i\}$ of points,

  $$T\left(\sum_i a_i p_i\right) = \sum_i a_i T(p_i) \quad \text{where} \quad \sum_i a_i = 1$$

  (i.e., affine transformations ***commute*** with affine combinations).
- Affine transformations in $\mathbb{R}^n$ ***preserve barycentric coordinates*** (which trivially follows from the previous property).

Homogeneous Coordinates

- A point in $\mathbb{R}^3$ can be represented in many ways (e.g., Cartesian coordinates, spherical coordinates, cylindrical coordinates).
- Which representation is most convenient depends on the application at hand.
- Homogeneous coordinates provide yet another way to represent a point in $\mathbb{R}^3$.
- The **homogeneous coordinates** of a point $p = (p_\mathsf{x}, p_\mathsf{y}, p_\mathsf{z})$ in $\mathbb{R}^3$ is a 4-tuple $(q_\mathsf{x}, q_\mathsf{y}, q_\mathsf{z}, q_\mathsf{w})$ satisfying $q_\mathsf{w} \neq 0$, and $p_\mathsf{x} = q_\mathsf{x}/q_\mathsf{w}$, $p_\mathsf{y} = q_\mathsf{y}/q_\mathsf{w}$, and $p_\mathsf{z} = q_\mathsf{z}/q_\mathsf{w}$.
- The homogeneous coordinates of a point are ***not unique***. If $(p_\mathsf{x}, p_\mathsf{y}, p_\mathsf{z}, p_\mathsf{w})$ is the homogeneous coordinates of a point, then so too is $(kp_\mathsf{x}, kp_\mathsf{y}, kp_\mathsf{z}, kp_\mathsf{w})$ for any real $k \neq 0$.
- Two homogeneous coordinates represent same point if and only if one point is a multiple of other.

Homogeneous-Coordinate Transformations

- When Cartesian coordinates are used along with 3×3 transformation matrices, translations and perspective projections are ***problematic*** as they have no corresponding matrix representation.
- The situation is quite different with homogeneous coordinates.
- As a matter of terminology, a transformation that operates on points expressed in homogeneous coordinates is referred to as **homogeneous-coordinate transformation**.
- Since homogeneous coordinates are a 4-tuple, homogeneous-coordinate transformations are associated with 4×4 matrices.
- As it turns out, every affine transformation (including translations), orthographic/perspective projection, or composition thereof, can be represented by a homogeneous-coordinate transformation matrix.
- The main benefit of the homogeneous representation is ***uniformity***.
- All transformations of interest can be characterized by a matrix and the application/composition of transformations is achieved by matrix multiplication.

Translation

- The homogeneous-coordinate transformation matrix $T(d)$ that corresponds to a translation by $d = (d_\mathsf{x}, d_\mathsf{y}, d_\mathsf{z})$ is given by

$$T(d) = \begin{bmatrix} 1 & 0 & 0 & d_\mathsf{x} \\ 0 & 1 & 0 & d_\mathsf{y} \\ 0 & 0 & 1 & d_\mathsf{z} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- Applying the above transformation to the point p, we obtain:

$$T(d)\begin{bmatrix} p_\mathsf{x} \\ p_\mathsf{y} \\ p_\mathsf{z} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & d_\mathsf{x} \\ 0 & 1 & 0 & d_\mathsf{y} \\ 0 & 0 & 1 & d_\mathsf{z} \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} p_\mathsf{x} \\ p_\mathsf{y} \\ p_\mathsf{z} \\ 1 \end{bmatrix} = \begin{bmatrix} p_\mathsf{x} + d_\mathsf{x} \\ p_\mathsf{y} + d_\mathsf{y} \\ p_\mathsf{z} + d_\mathsf{z} \\ 1 \end{bmatrix}.$$

Scaling

- The homogeneous-coordinate transformation matrix $T(s)$, with $s = (s_x, s_y, s_z)$, that corresponds to a scaling in the x, y, and z directions by s_x, s_y, and s_z, respectively, is given by

$$S(s) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- Applying the above transformation to the point $p = (p_x, p_y, p_z)$, we obtain:

$$S(s) \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} s_x p_x \\ s_y p_y \\ s_z p_z \\ 1 \end{bmatrix}.$$

Rotation About z Axis

- The homogeneous-coordinate transformation matrix $R_{\mathrm{z}}(\theta)$ that corresponds to a rotation of θ about the z axis is given by

$$R_{\mathrm{z}}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- Applying the above transformation to the point $(p_{\mathrm{x}}, p_{\mathrm{y}}, p_{\mathrm{z}})$, we obtain:

$$R_{\mathrm{z}}(\theta) \begin{bmatrix} p_{\mathrm{x}} \\ p_{\mathrm{y}} \\ p_{\mathrm{z}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_{\mathrm{x}} \\ p_{\mathrm{y}} \\ p_{\mathrm{z}} \\ 1 \end{bmatrix} = \begin{bmatrix} p_{\mathrm{x}}\cos\theta - p_{\mathrm{y}}\sin\theta \\ p_{\mathrm{x}}\sin\theta + p_{\mathrm{y}}\cos\theta \\ p_{\mathrm{z}} \\ 1 \end{bmatrix}.$$

Rotation About x Axis

- The homogeneous-coordinate transformation matrix $R_{\mathrm{x}}(\theta)$ that corresponds to a rotation of θ about the x axis is given by

$$R_{\mathrm{x}}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- Applying the above transformation to the point $(p_{\mathrm{x}}, p_{\mathrm{y}}, p_{\mathrm{z}})$, we obtain:

$$R_{\mathrm{x}}(\theta) \begin{bmatrix} p_{\mathrm{x}} \\ p_{\mathrm{y}} \\ p_{\mathrm{z}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_{\mathrm{x}} \\ p_{\mathrm{y}} \\ p_{\mathrm{z}} \\ 1 \end{bmatrix} = \begin{bmatrix} p_{\mathrm{x}} \\ p_{\mathrm{y}}\cos\theta - p_{\mathrm{z}}\sin\theta \\ p_{\mathrm{y}}\sin\theta + p_{\mathrm{z}}\cos\theta \\ 1 \end{bmatrix}.$$

Rotation About y Axis

- The homogeneous-coordinate transformation matrix $R_{\mathrm{y}}(\theta)$ that corresponds to a rotation of θ about the y axis is given by

$$R_{\mathrm{y}}(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- Applying the above transformation to the point $(p_{\mathrm{x}}, p_{\mathrm{y}}, p_{\mathrm{z}})$, we obtain:

$$R_{\mathrm{y}}(\theta)\begin{bmatrix} p_{\mathrm{x}} \\ p_{\mathrm{y}} \\ p_{\mathrm{z}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} p_{\mathrm{x}} \\ p_{\mathrm{y}} \\ p_{\mathrm{z}} \\ 1 \end{bmatrix} = \begin{bmatrix} p_{\mathrm{x}}\cos\theta + p_{\mathrm{z}}\sin\theta \\ p_{\mathrm{y}} \\ -p_{\mathrm{x}}\sin\theta + p_{\mathrm{z}}\cos\theta \\ 1 \end{bmatrix}.$$

Rotation About Arbitrary Axis

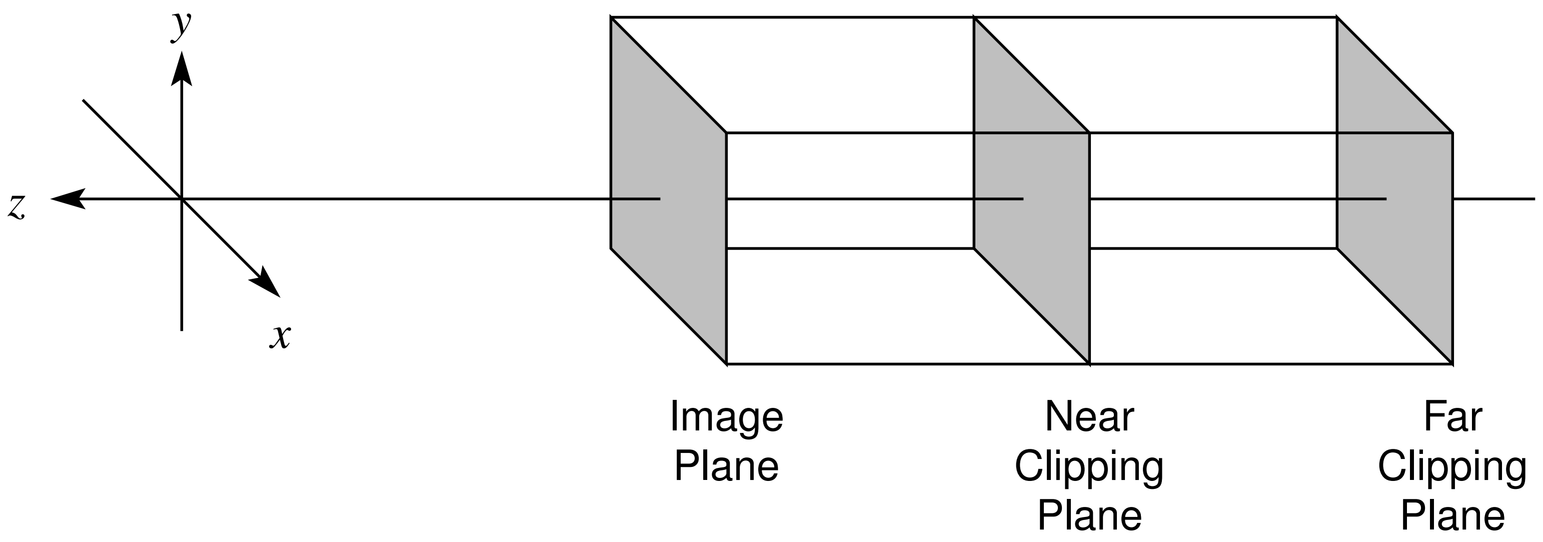
- The homogeneous-coordinate transformation matrix $R(a,\theta)$ that corresponds to a rotation of θ about the axis in the direction of the unit vector $a = (a_\mathsf{x}, a_\mathsf{y}, a_\mathsf{z})$ is given by

$$R(a,\theta) = \begin{bmatrix} a_\mathsf{x}^2 + c_\theta(1-a_\mathsf{x}^2) & a_\mathsf{x}a_\mathsf{y}(1-c_\theta) - a_\mathsf{z}s_\theta & a_\mathsf{x}a_\mathsf{z}(1-c_\theta) + a_\mathsf{y}s_\theta & 0 \\ a_\mathsf{x}a_\mathsf{y}(1-c_\theta) + a_\mathsf{z}s_\theta & a_\mathsf{y}^2 + c_\theta(1-a_\mathsf{y}^2) & a_\mathsf{y}a_\mathsf{z}(1-c_\theta) - a_\mathsf{x}s_\theta & 0 \\ a_\mathsf{x}a_\mathsf{z}(1-c_\theta) - a_\mathsf{y}s_\theta & a_\mathsf{y}a_\mathsf{z}(1-c_\theta) + a_\mathsf{x}s_\theta & a_\mathsf{z}^2 + c_\theta(1-a_\mathsf{z}^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  where $c_\theta = \cos\theta$ and $s_\theta = \sin\theta$.

- The composition of any number of rotations can always be represented as a single rotation about an arbitrary axis.

Orthographic Projection



- projection lines orthogonal to projection plane (i.e., image plane)
- effectively, as if eye is on positive z axis infinitely far from origin, looking in negative z direction with positive y axis corresponding to up direction
- viewing volume is rectangular prism (bounded by near and far clipping planes)
- line segments of equal length have projections of equal length

Orthographic Projection (Continued)

- The homogeneous-coordinate transformation matrix P that corresponds to an orthographic projection onto the image plane $z = 0$ is given by

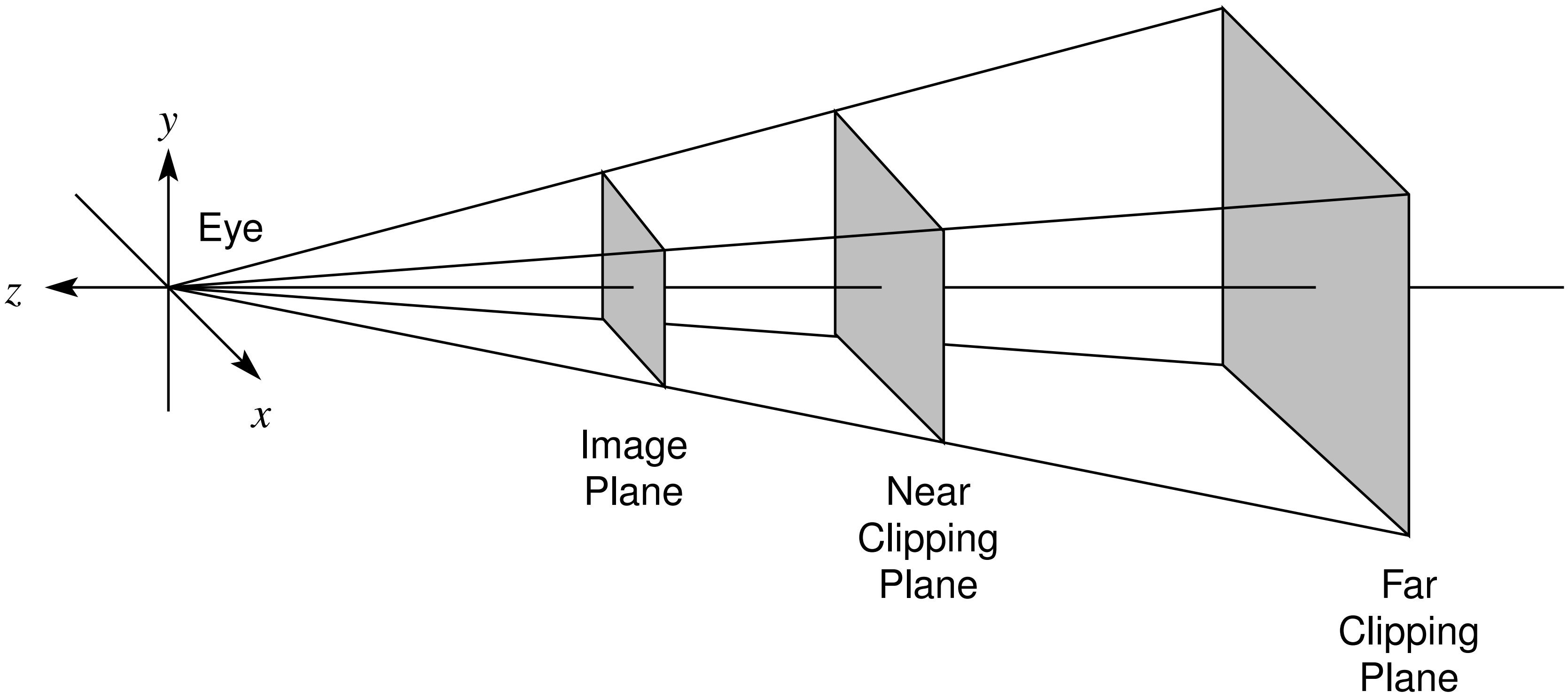
$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- Applying the above transformation to the point $(p_\mathrm{x}, p_\mathrm{y}, p_\mathrm{z})$, we obtain:

$$P \begin{bmatrix} p_\mathrm{x} \\ p_\mathrm{y} \\ p_\mathrm{z} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_\mathrm{x} \\ p_\mathrm{y} \\ p_\mathrm{z} \\ 1 \end{bmatrix} = \begin{bmatrix} p_\mathrm{x} \\ p_\mathrm{y} \\ 0 \\ 1 \end{bmatrix}.$$

- More complicated orthographic projections can be constructed by combining the above transformation with scaling, rotation, and translation to move the image plane and center of projection wherever they are desired.

Perspective Projection



- all projection lines converge to single point at eye
- eye is located at origin looking in negative z direction with positive y axis corresponding to up direction
- viewing volume is frustum (bounded by near and far clipping planes)
- given two identical objects, one closer to eye appears larger

Perspective Projection (Continued)

- The homogeneous-coordinate transformation matrix P that corresponds to a perspective projection, with the origin as the center of the projection and $z = 1$ as the image plane, is given by

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

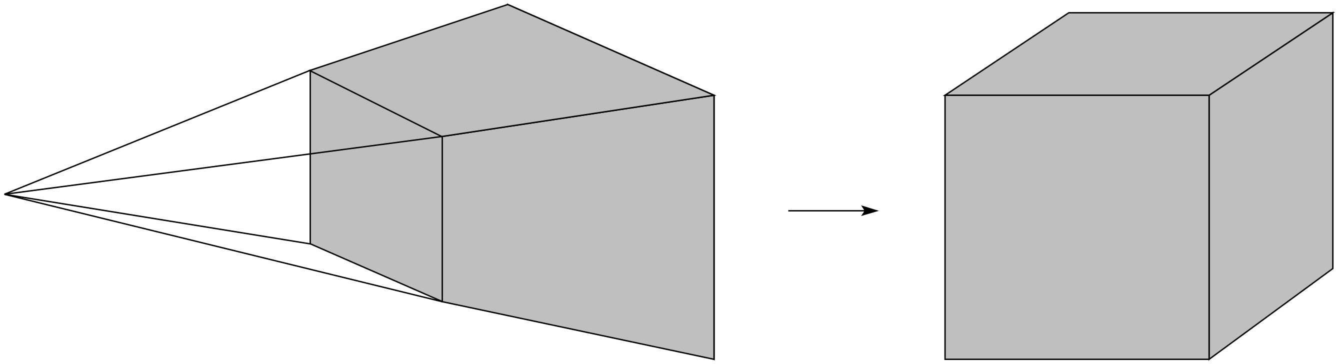
- Applying the above transformation to the point $(p_\mathsf{x}, p_\mathsf{y}, p_\mathsf{z})$, we obtain:

$$P \begin{bmatrix} p_\mathsf{x} \\ p_\mathsf{y} \\ p_\mathsf{z} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_\mathsf{x} \\ p_\mathsf{y} \\ p_\mathsf{z} \\ 1 \end{bmatrix} = \begin{bmatrix} p_\mathsf{x} \\ p_\mathsf{y} \\ p_\mathsf{z} \\ p_\mathsf{z} \end{bmatrix} \sim \begin{bmatrix} p_\mathsf{x}/p_\mathsf{z} \\ p_\mathsf{y}/p_\mathsf{z} \\ 1 \\ 1 \end{bmatrix}.$$

- More complicated perspective projections can be constructed by combining the above transformation with scaling, rotation, and translation to move the image plane and center of projection wherever they are desired.

Perspective Projection as Warping Plus Orthographic Projection

- Perspective projection can always be decomposed into two separate transformations, namely, a warping followed by orthographic projection (each of which has a homogeneous-coordinate transformation matrix representation).
- The warping maps the viewing frustum associated with perspective projection into a cube as shown below.



Transformations As Change in Coordinate System

- For a transformation matrix T and a point p expressed as a column vector, consider the product Tp.
- The product Tp can be interpreted in two distinct but equivalent ways:
  1. As the ***transformation of a point***: The product Tp is the new point produced by applying the transformation T to the point p.
  2. As the ***transformation of a coordinate system***: The product Tp is the new point obtained by applying the transformation T^{-1} to the coordinate-system axes and then interpreting p relative to these new coordinate-system axes.
- Although the first interpretation is, perhaps, the most straightforward, the second interpretation is often very useful in computer graphics applications.
- Note: In the case that T above is a composite transformation $T = T_n T_{n-1} \cdots T_1$, recall that $T^{-1} = T_1^{-1} T_2^{-1} \cdots T_n^{-1}$.

Example: Transformation

- consider transformation T given by

$$T = R_z(90^\circ) S\left(\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}\right) T(1,-1,0)$$

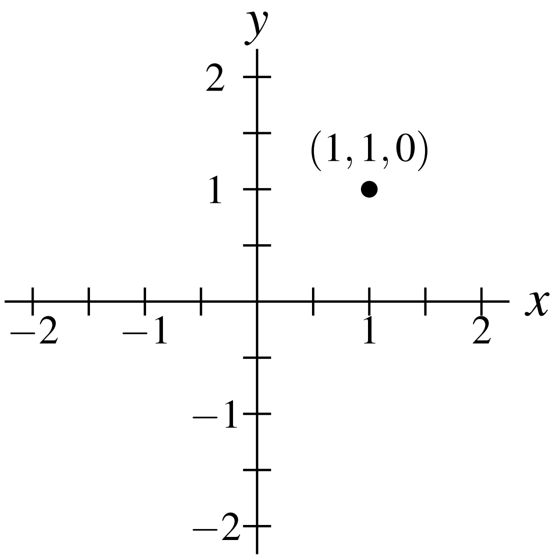
$$= \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- consider point $(1,1,0)$ with homogeneous coordinate representation given by

$$p = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

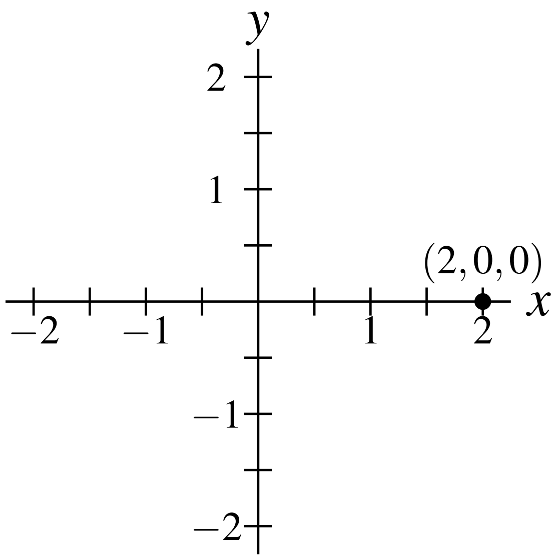
- two different interpretations of product Tp as shown on next two slides

Example: Transforming Point



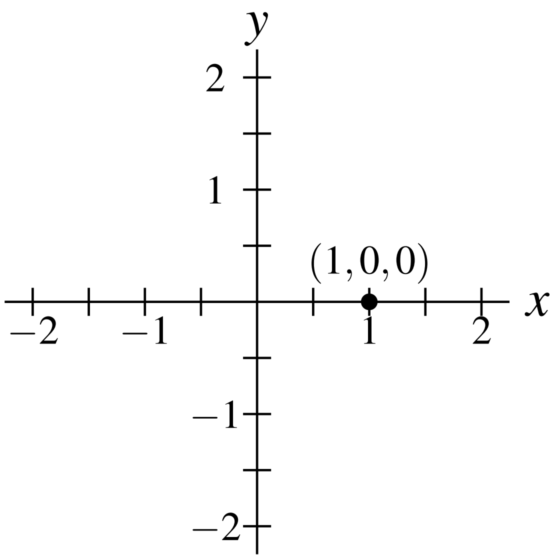
$$\xrightarrow{T(1,-1,0)}$$

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$



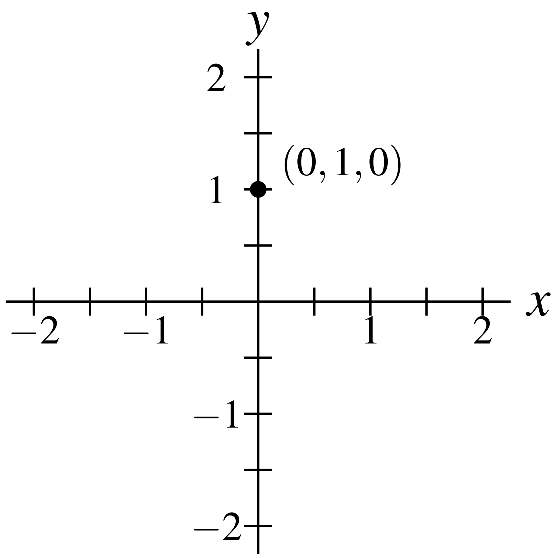
$$\xrightarrow{S\left(\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)}$$

$$\begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$



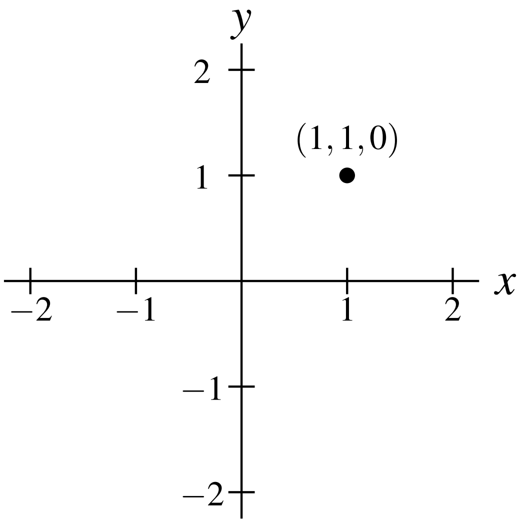
$$\xrightarrow{R_z(90^\circ)}$$

$$\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

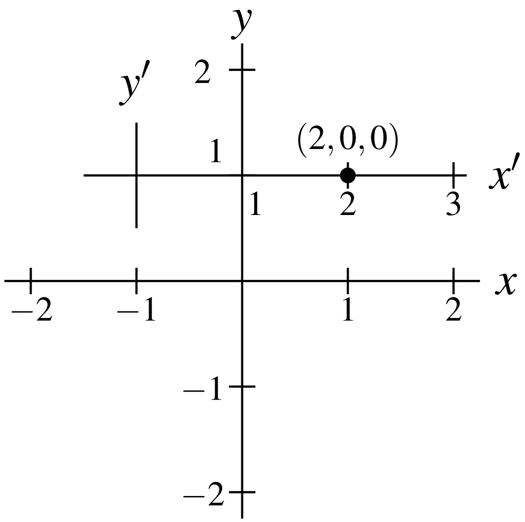


The z axis is not explicitly shown and is coming out of the plane of the page.

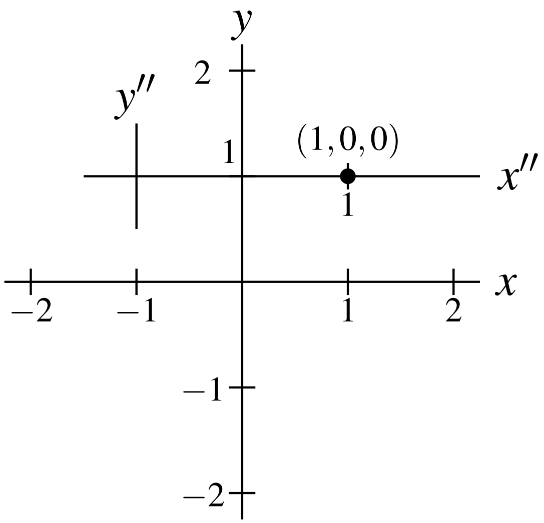
Example: Transforming Coordinate System



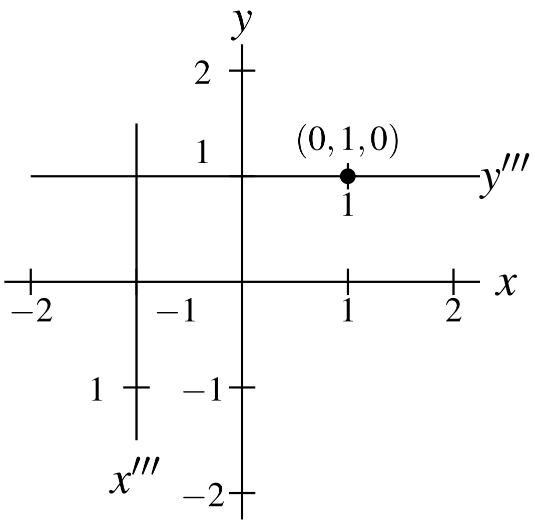
translate by $(-1,1,0)$ $\longrightarrow$



scale by $(2,2,2)$ $\longrightarrow$



rotate about z-axis by -90° $\longrightarrow$



The z axis is not explicitly shown and is coming out of the plane of the page.

Section 5.2

Quaternions

Quaternions

- The quaternions, denoted $\mathbb{H}$, can be viewed as an extension of complex numbers, and were discovered by William Rowan Hamilton in 1843.
- Quaternions are extremely useful for representing ***rotations in 3-D***, and have application in areas such as ***computer graphics*** and robotics.
- A quaternion q has the form $q = w + xi + yj + zk$ where $w, x, y, z \in \mathbb{R}$ and $i^2 = j^2 = k^2 = ijk = -1$.
- A quaternion $q = w + xi + yj + zk$ can be viewed as consisting of a **scalar part**, namely w, and a **vector part** with components in the i, j, and k directions, namely (x, y, z).
- The quaternion $q = w + xi + yj + zk$ with the scalar part $s = w$ and vector part $v = (x, y, z)$ is denoted as (s, v).
- The conjugate of $q = (s, v)$, denoted q^*, is defined as $q^* = (s, -v)$.
- The norm of q is defined as $\|q\| = \sqrt{qq^*}$.

Quaternions (Continued)

- For quaternions, addition is defined as $(s_1, v_1) + (s_2, v_2) = (s_1 + s_2, v_1 + v_2)$.
- For quaternions, multiplication is defined as $(s_1, v_1)(s_2, v_2) = (s_1 s_2 - v_1 \cdot v_2, s_1 v_2 + s_2 v_1 + v_1 \times v_2)$.
- The multiplicative inverse of q, denoted q^{-1}, is given by $q^* / \|q\|^2$.
- Addition is commutative. Multiplication is ***not commutative***.
- A rotation by the angle θ about the axis v (using a right-hand rule) can be represented by the unit-norm quaternion $(\cos\theta/2, \frac{1}{\|v\|} v \sin\theta/2)$.
- Quaternion multiplication then corresponds to rotation matrix multiplication. Let q_1 and q_2 be quaternions associated with rotations R_1 and R_2, respectively. Then, the quaternion product $q_2 q_1$ corresponds to the rotation $R_2 R_1$ (i.e., R_1 followed by R_2).
- Let v_1 and v_2 denote points on the 3-D unit sphere. The quaternion quotient $(0, v_2)/(0, v_1)$ corresponds to a rotation along the great circle arc from v_1 to v_2.

Section 5.3

Manifolds

Topological Spaces

- A **topology** on a set X is a set T of subsets of X satisfying:
  1. Both $\emptyset$ and X belong to T.
  2. The union of ***any*** collection of sets from T also belongs to T.
  3. The intersection of any ***finite*** collection of sets from T belongs to T.

  The sets in T will be called the **open** sets of X.
- Basically, a topology on a set X specifies which points in X are close to one another (i.e., which points are neighbours).
- A **topological space** is a set X with a topology T (on X), and is denoted as (X, T), or simply X when T is clear from the context.
- A metric space is a topological space, since a metric induces a topology on a set.
- Not all topologies can be described in terms of a metric, however. So, not all topological spaces are metric spaces.

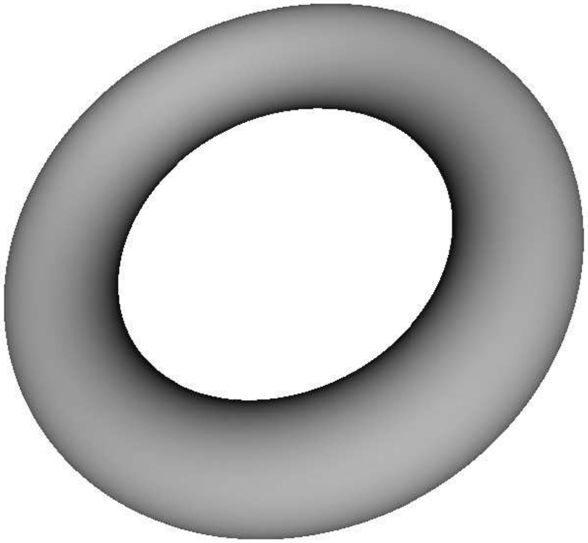
Topological Spaces (Continued)

- In the case of a metric space, a metric serves as the starting point for defining a topology on a set.
- In the more general topological-space setting, the topology on a set X is specified directly by identifying all open subsets of X. Then, closed sets and neighbourhoods are defined in terms of open sets. Other concepts (such as closure, boundary, and interior) then follow.
- A subset S of a topological space X is said to be **closed** if $X \setminus S$ is open.
- A subset S of a topological space X is said to be a **neighbourhood** of a point $p \in X$ if there exists an open set U such that $p \in U \subset S$ (i.e., S contains an open set that contains the point p).
- A topological space X is said to be a **Hausdorff space** if any two distinct points of X have disjoint neighbourhoods.
- All metric spaces are Hausdorff spaces. Not every topological space is a Hausdorff space, however.

Homeomorphisms

- A mapping $T : X \to Y$ that is one-to-one and onto, where T and T^{-1} are both continuous, is called a **homeomorphism**.
- A homeomorphism can be thought of as an ***elastic deformation*** (i.e., stretching/compressing, bending/twisting, but no cutting/tearing).
- Two spaces X and Y are said to be **homeomorphic** is there exists a homeomorphism $T : X \to Y$.
- Two topological spaces X and Y are equivalent, denoted $X = Y$, if they are homeomorphic.

Homeomorphism Example: Coffee Cup and Donut

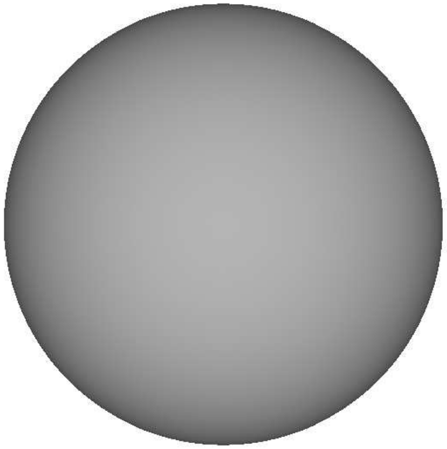


- Let C and D denote the surfaces of the coffee cup and donut, respectively (as shown above).
- Since C can be transformed into D (or vice versa) by an elastic deformation, C and D are homeomorphic.
- Thus, $C = D$ (i.e., a coffee cup and donut are topologically equivalent).

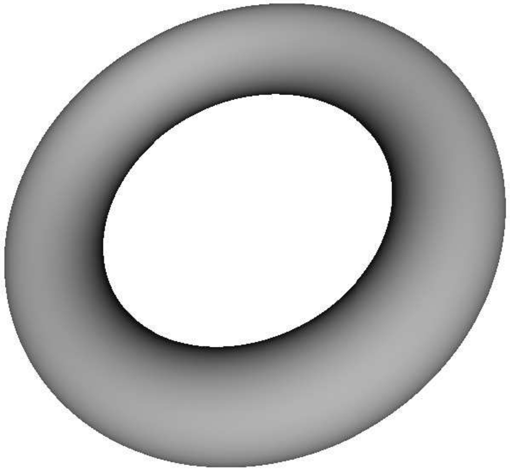
Manifolds

- A **n-dimensional manifold** (called an n-manifold) is a Hausdorff space M such that each point $p \in M$ has a neighbourhood homeomorphic to the open n-dimensional unit disc $U^n = \{(x_1, x_2, \ldots, x_n) \in \mathbb{R}^n : \sum_{k=1}^{n} x_k^2 < 1\}$.
- A **n-dimensional manifold with boundary** (called an n-manifold with boundary) is a Hausdorff space M such that each point $p \in M$ has a neighbourhood homeomorphic to either the open n-dimensional unit disc U^n or the open n-dimensional unit half-disc $U^n_+ = \{(x_1, x_2, \ldots, x_n) : \sum_{k=1}^{n} x_k^2 < 1$ and $x_1 \geq 0\}$.
- A 2-manifold is called a **surface**.
- An n-manifold (with or without boundary) locally has the same properties as $\mathbb{R}^n$. For example, an infinitesimally small bug crawling along a 2-manifold M could not distinguish M from the plane $\mathbb{R}^2$ if the range of the bug's vision were restricted to only its local neighbourhood.
- Examples of 1-manifolds: a line, a circle
- An example of an n-manifold is $\mathbb{R}^n$.

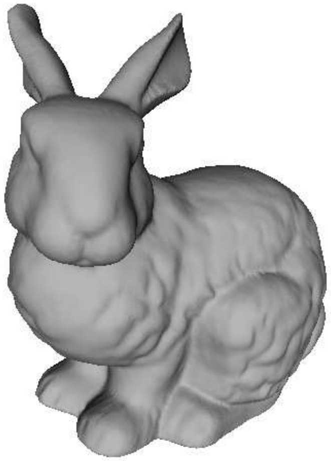
Examples of 2-Manifolds With/Without Boundaries



Sphere

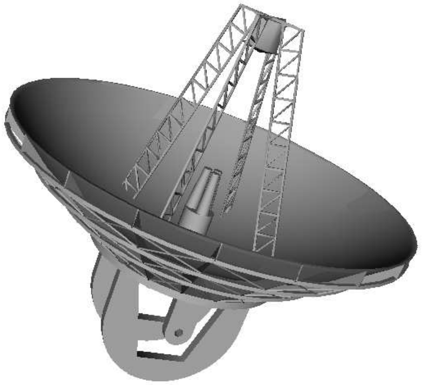


Torus

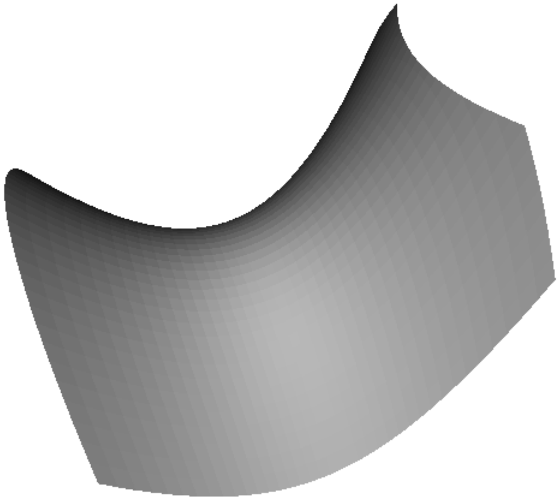


Surface of Rabbit

Surface of Spacesuit

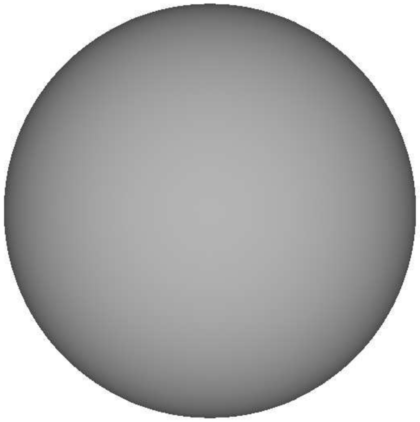


Surface of Telescope Dish



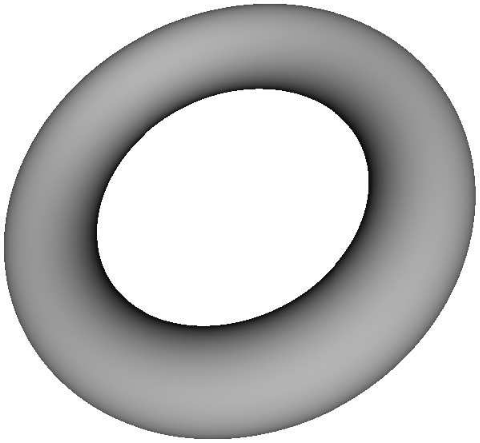
Simple Surface

Examples of 3-Manifolds With Boundaries



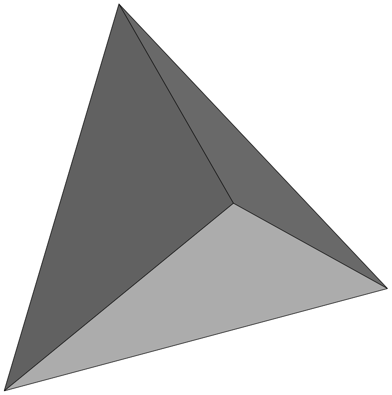
Ball

(Sphere and Its Interior)



Toroid

(Torus and Its Interior)

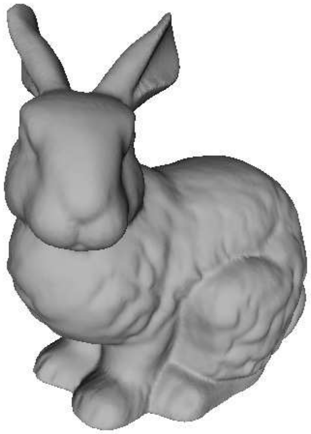


Tetrahedron

and Its Interior

Spacesuit

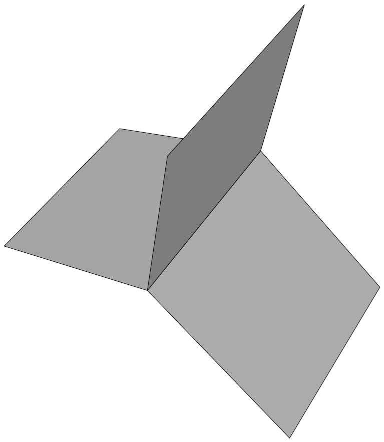
(Including Its Interior)



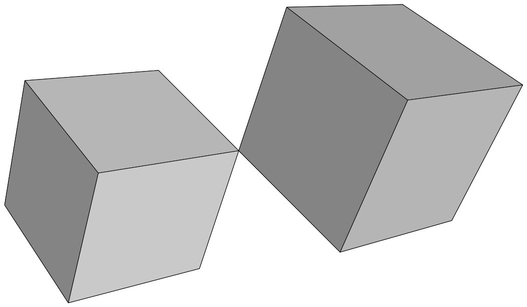
Rabbit

(Including Its Interior)

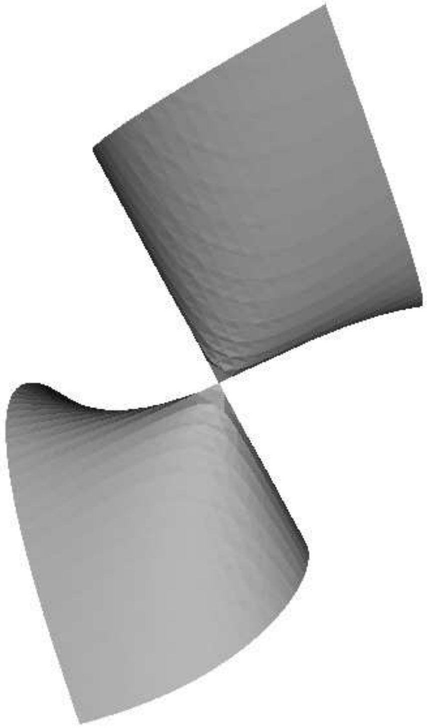
Examples of Non-Manifolds



Three Quadrilaterals Intersecting At Common Edge



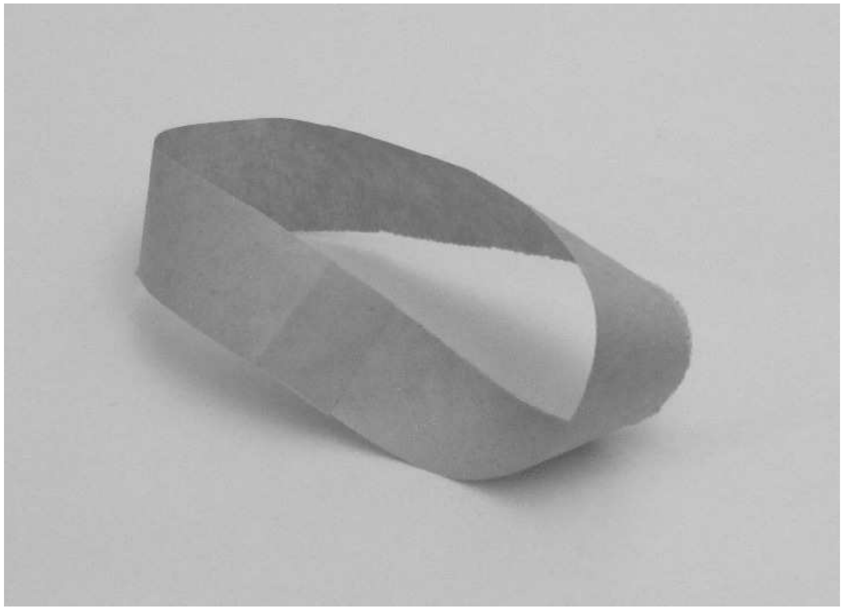
Two Cubes Intersecting at Common Vertex



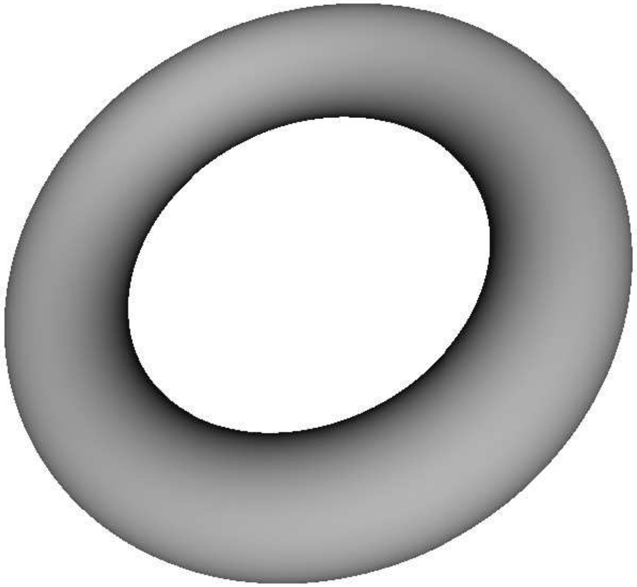
Two Surfaces Intersecting at a Point

Orientability

- A manifold is said to be **orientable** if one can consistently define a clockwise direction for all loops in the manifold.
- We are interested in the orientability of surfaces (i.e., 2-manifolds).
- Most surfaces encountered in the physical world are orientable.
- Examples of orientable surfaces: a sphere, plane, torus, and polyhedron.
- Example of a non-orientable surface: a Mobius strip.



Mobius Strip (Non-Orientable)



Torus (Orientable)

Signals in Geometry Processing

- The signals dealt with in geometry processing are most commonly manifolds.
- In traditional signal processing, a signal is a function, which is a ***vector*** in an inner-product space.
- In geometry processing, however, a signal is a manifold, which is a (topological) ***space***.
- In most practical applications, the n-manifold M (with or without boundary) of interest is imbedded into a Euclidean space $\mathbb{R}^d$, where $n \leq d$ (i.e., $M \subset \mathbb{R}^d$).
- Generally speaking, 2- and 3-manifolds (with or without boundaries) tend to be of interest most frequently.
- As one might expect, in the context of subdivision surfaces, we are interested in surfaces (i.e., 2-manifolds) with or without boundaries.

Representations of Surfaces

- To facilitate the processing of surfaces (i.e., 2-manifolds), we need a way to define/represent them.
- A surface can be defined ***implicitly***. That is, a surface can be defined to consist of all of the points satisfying a particular equation.
  E.g., the surface of a sphere with radius ρ and center at the origin consists of the points $\{(x,y,z) \in \mathbb{R}^3 : x^2 + y^2 + z^2 = \rho^2\}$.
- A surface can be defined ***parametrically***. That is, a surface can be defined to consist of the points in the range of a vector-valued function (i.e., $p(u,v) = (f_x(u,v), f_y(u,v), f_z(u,v))$.
  E.g., the surface of a sphere with radius ρ and center at the origin consists of the points in the range of the function
  $p(u,v) = (\rho\cos(u)\cos(v), \rho\sin(u)\cos(v), \rho\sin(v))$, where
  $u \in [0, 2\pi), v \in [0, \pi]$.
- Often, implicit and parametric definitions are not the most convenient to utilize.
- Yet another approach is to define a surface (or an approximation thereof) using a ***polygon mesh***.

Parametric Continuity

- Parametric continuity is a measure of the ***smoothness*** of a surface.
- The parametric form of a surface S (imbedded in $\mathbb{R}^3$) is given by $p(u,v) = \begin{bmatrix} x(u,v) & y(u,v) & z(u,v) \end{bmatrix}^T$.
- The surface S is traced out by p as the parameters u and v varied over their domain.
- The surface S is **C^n continuous** if all partial derivatives of p of order n or less exist and are continuous.
- A tangent plane to S is determined by the first-order partial derivatives $\frac{\partial p(u,v)}{\partial u} = \begin{bmatrix} \frac{\partial x(u,v)}{\partial u} & \frac{\partial y(u,v)}{\partial u} & \frac{\partial z(u,v)}{\partial u} \end{bmatrix}^T$ and $\frac{\partial p(u,v)}{\partial v} = \begin{bmatrix} \frac{\partial x(u,v)}{\partial v} & \frac{\partial y(u,v)}{\partial v} & \frac{\partial z(u,v)}{\partial v} \end{bmatrix}^T$.
- The parameterization of a surface is ***not unique***.
- Different parameterizations of the same surface can have different parametric continuity. That is, parametric continuity depends not only on the surface being parameterized, but also on the ***parameterization itself***.

Geometric Continuity

- Geometric continuity is a measure of the ***smoothness*** of a surface that is ***independent*** of any surface parameterization.
- A surface is **G^0 continuous** if it does not have any jumps (i.e., is continuous).
- A surface is **G^1 continuous** if it has a continuously varying tangent plane.
- A surface is **G^2 continuous** if it has continuously varying curvature.
- A polyhedral surface is G^0 continuous but (typically) not G^1 continuous (since the tangent plane does not vary continuously between faces of a polyhedron).
- Parametric continuity is a stronger form of continuity than geometric continuity. That is, C^n continuity implies G^n continuity (but G^n continuity does not imply C^n continuity).

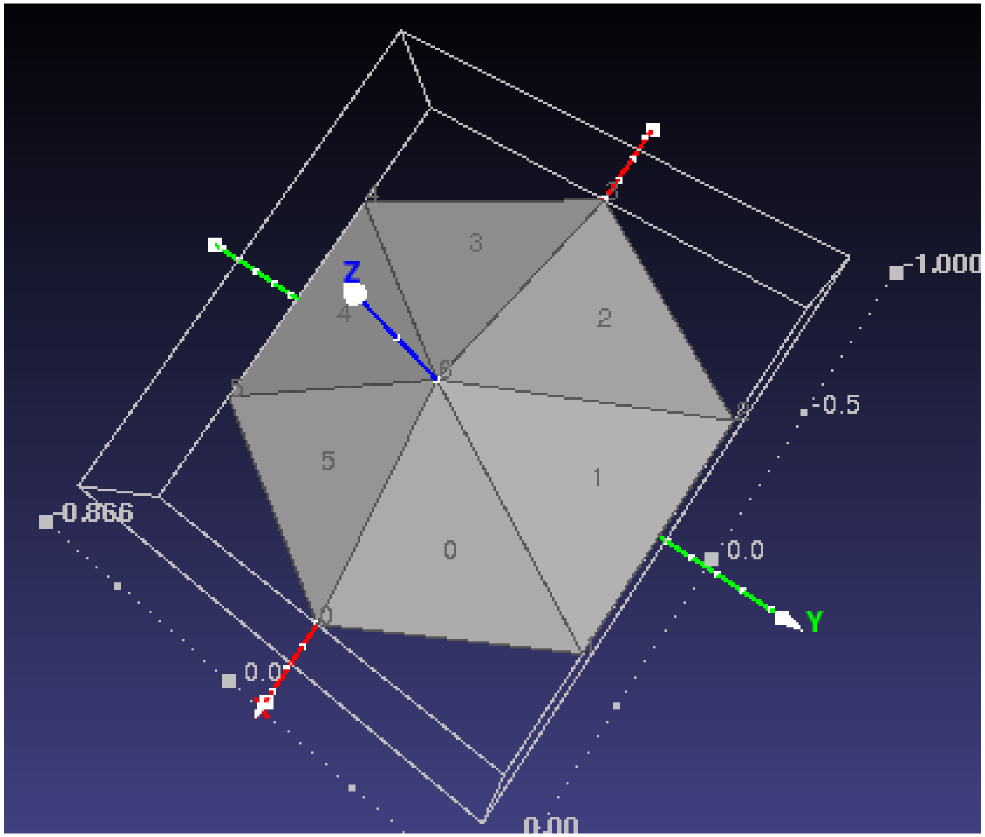
Section 5.4

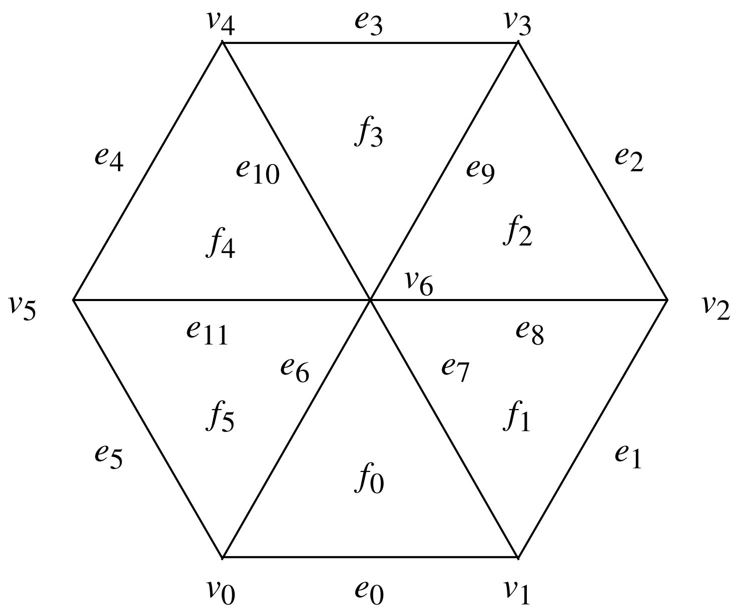
Polygon Meshes

Polygon Meshes

- A **polygon mesh** is a collection of vertices, edges, and (polygonal) faces, and an incidence relationship amongst them. An edge connects two vertices, and a face is a closed sequence of edges.
- A polygon mesh consists of two types of information:
  1. ***geometric information:*** the positions of the vertices (in the Euclidean space in which the mesh is imbedded); and
  2. ***topologic information***: how the vertices are connected together to form edges and faces (i.e., the connectivity of the mesh).
- In practice, the faces in a mesh are most commonly all triangles, all quadrilaterals, or a mixture of triangles and quadrilaterals.
- A polygon mesh with all triangle faces is called a **triangle mesh**.
- A polygon mesh with all quadrilateral faces is called a **quadrilateral mesh** (or **quad mesh**).
- A polygon mesh can be manifold or non-manifold. Also, it can be orientable or non-orientable.

Polygon Mesh Example





for $k \in \{0,1,2,3,4,5\}$:

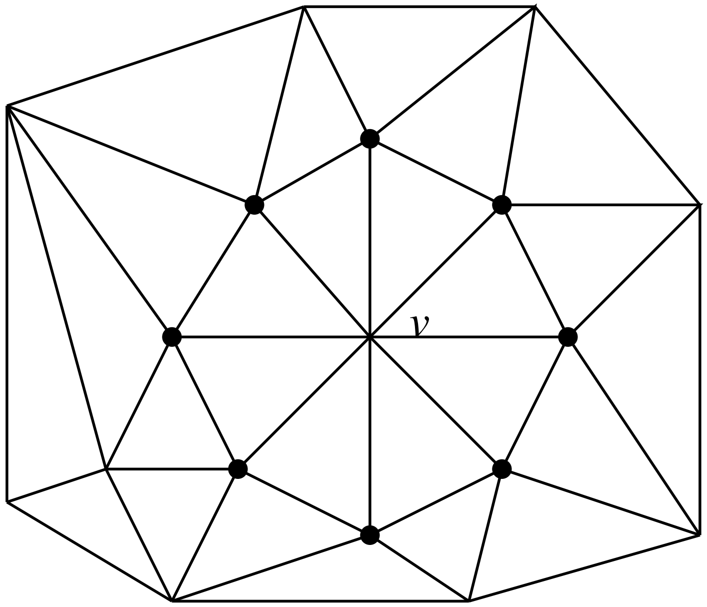
$$v_k = (\cos(\tfrac{\pi}{3}[k-2]), \sin(\tfrac{\pi}{3}[k-2]), 0)$$

$$v_6 = (0, 0, \tfrac{1}{2})$$

- A orientable manifold triangle mesh imbedded in $\mathbb{R}^3$
- Consists of 7 vertices, 12 edges, and 6 (triangle) faces
- E.g., face f_0 consists of vertices v_0, v_1, v_6 and edges e_0, e_7, e_6

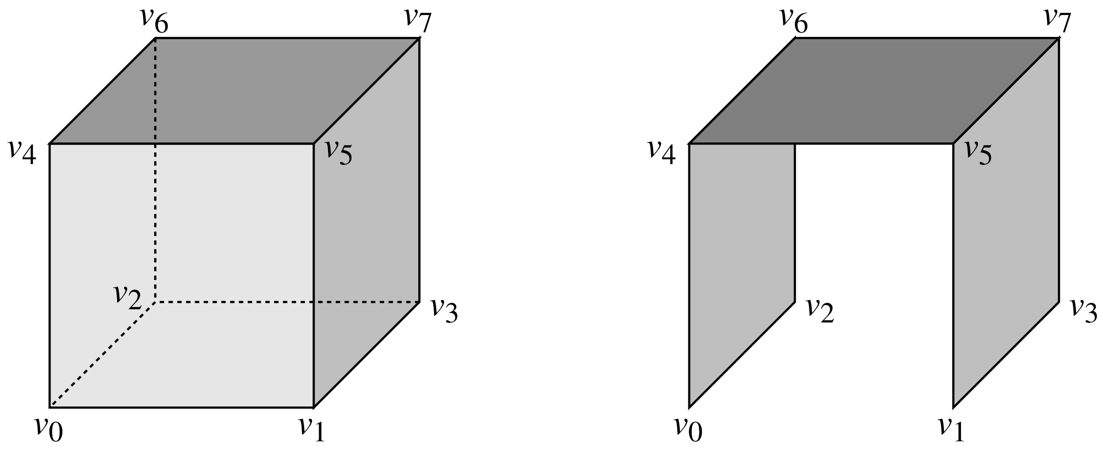
Valence and 1-Ring

- The **valence** of a vertex is the number of edges incident on that vertex.
- The **1-ring** of a vertex v is the set of all vertices in the mesh that are directly connected by an edge to v.

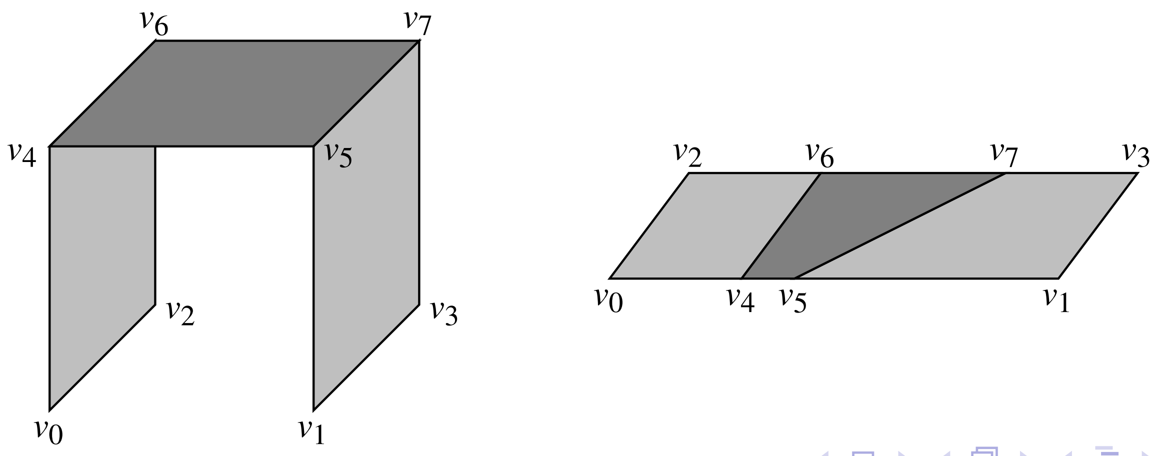


Topology Versus Geometry

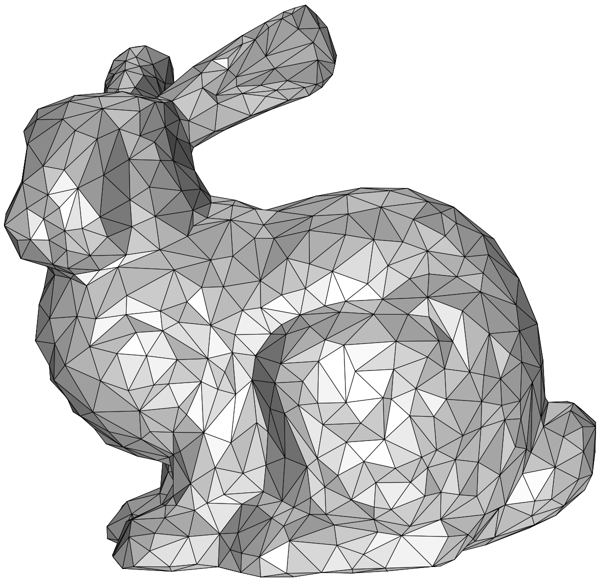
- The following is an example of two quadrilateral meshes with the same geometry but different topology:

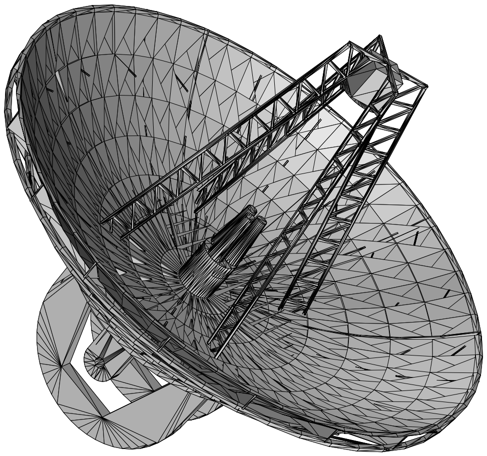


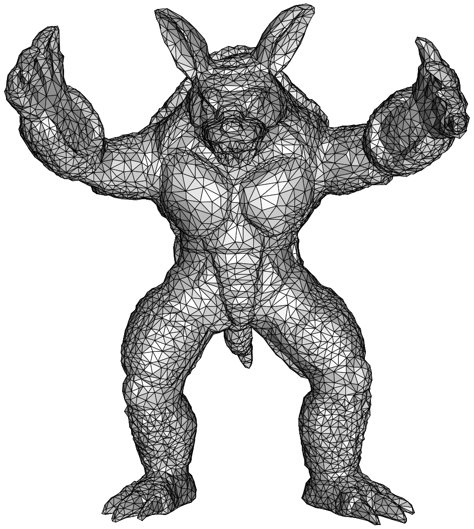
- The following is an example of two quadrilateral meshes with the same topology but different geometry:

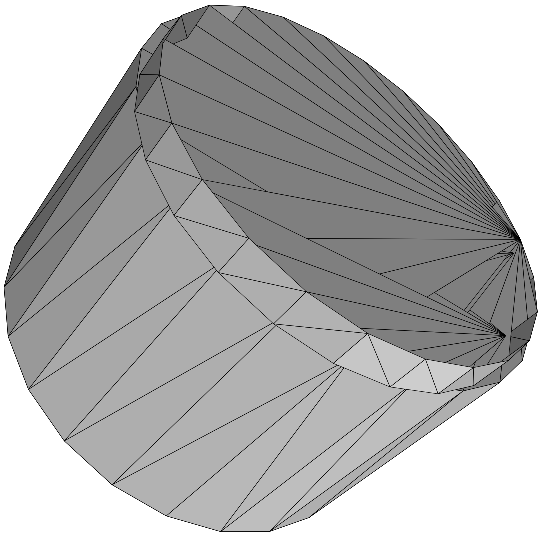


Examples of Polygon Meshes









Section 5.5

Data Structures and File Formats for Polygon Meshes

Naive Data Structure

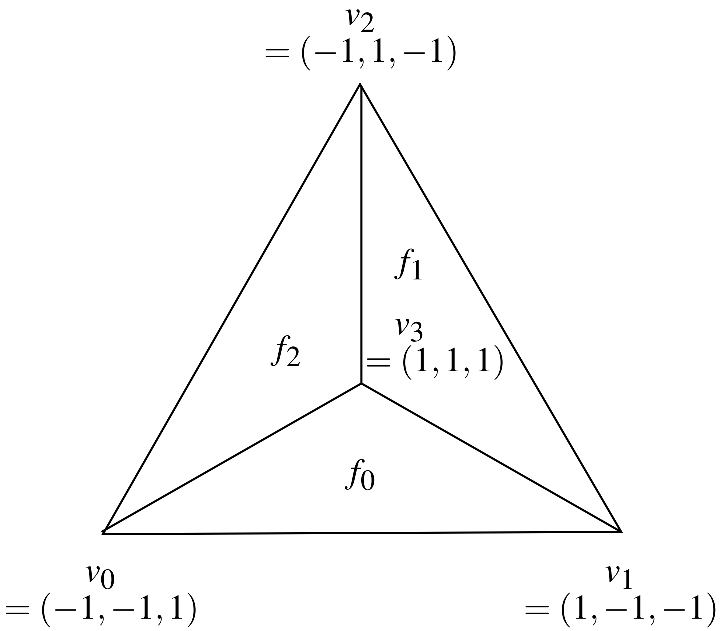
Pseudocode (Data-type definition)

```
Vertex vertices[numVertices]; // vertex array
Face triangles[numTriangles]; // triangle array

struct Face {
    int vertexIndexes[3]; // indexes of vertices for this triangle
};
```

- edges not explicitly represented
- some adjacency information not readily accessible
- to find neighboring face, must scan through face array looking for face with two vertices in common, which takes $O(n)$ time (where n is number of vertices)
- need data structures that allow efficient access to adjacency information

Naive Data Structure Example



Vertices

Array Index	Array Element
0	(-1,-1,1)
1	(1,-1,-1)
2	(-1,1,-1)
3	(1,1,1)

Faces

Array Index	Array Element
0	0, 1, 3
1	1, 2, 3
2	0, 3, 2

Winged-Edge Data Structure

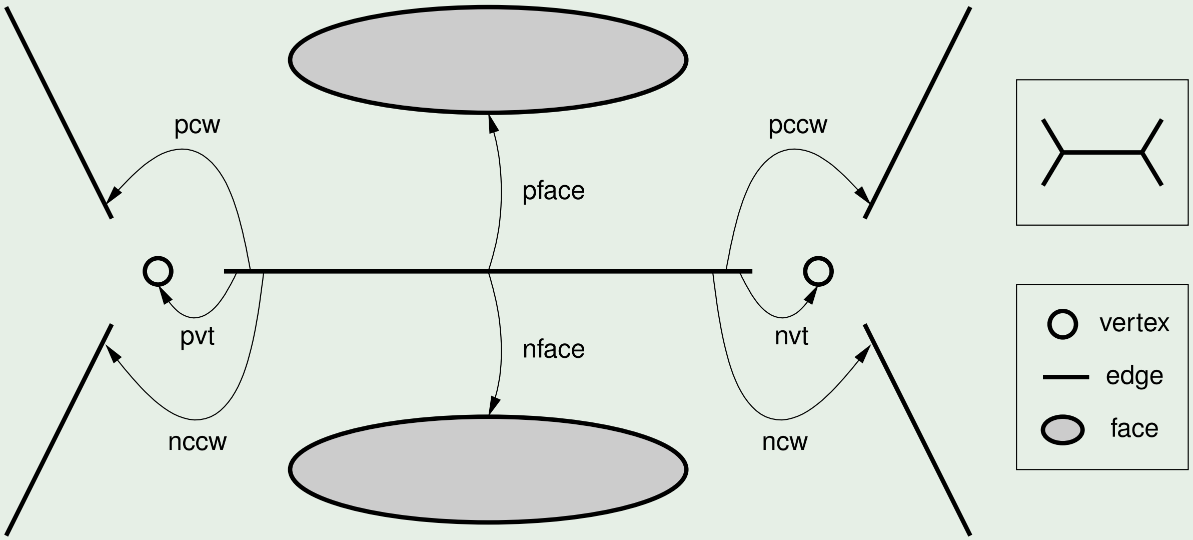
- proposed in:
  B. G. Baumgart. A polyhedron representation for computer vision. In *AFIPS National Computer Conference*, pages 589–596, 1975.
- edge-based representation, where edges serve as glue that holds vertices and faces together
- each edge points to two incident faces and two incident vertices and four incident edges that share same faces and vertices (i.e., four “wing” edges)
- since edges have no direction, traversing edges in particular direction (e.g., CCW/CW around face) requires one case distinction per step
- 8 pointers per edge

Winged-Edge Data Structure (Continued)

Pseudocode (Data-type definition)

```
struct Edge {
    Vertex* pvt; // first vertex
    Vertex* nvt; // second vertex
    Face* pface; // first face (defined to be to left of directed line
                 // segment from first to second vertex)
    Face* nface; // second face (defined as other face incident on edge)
    Edge* pccw;  // neighboring edge on first face in CCW direction
    Edge* pcw;   // neighboring edge on first face in CW direction
    Edge* nccw;  // neighboring edge on second face in CCW direction
    Edge* ncw;   // neighboring edge on second face in CW direction
};
```

Example (Pictorial view of data structure)



Half-Edge Data Structure

- described in:
  K. Weiler. Edge-based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, Jan. 1985.
- every edge represented as pair of directed edges, each called half-edge
- 6 pointers plus 2 bits (i.e., 2 one-bit integers) per edge
- used in Computational Geometry Algorithms Library (CGAL)
- representing edges in terms of *directed* line segments avoids some problems associated with winged-edge data structure

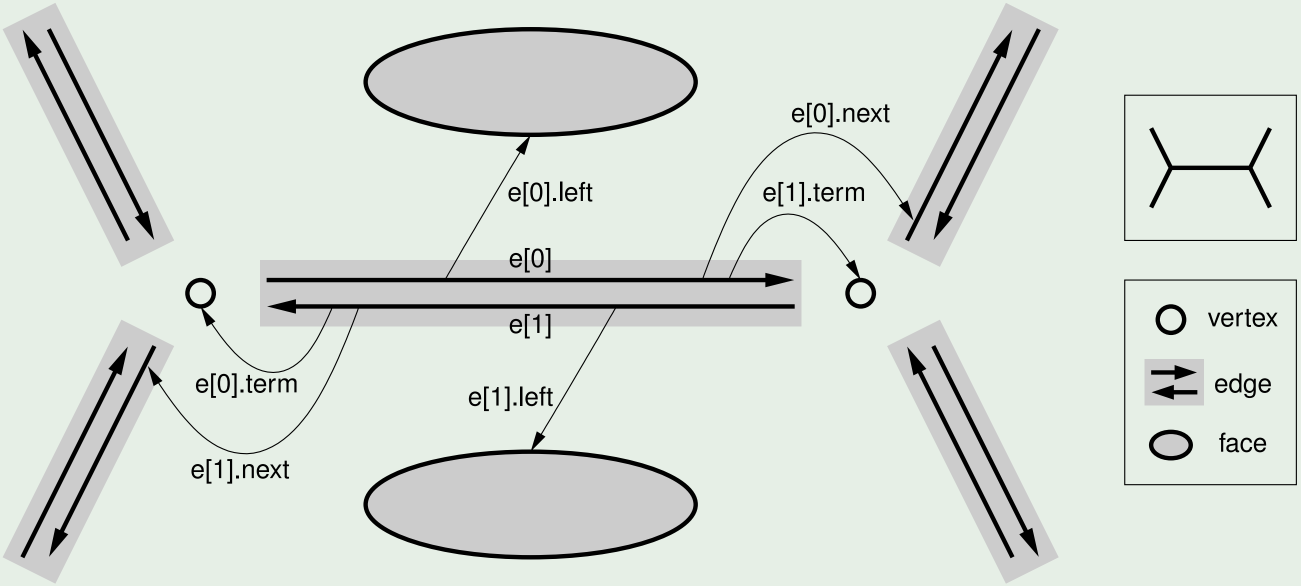
Half-Edge Data Structure (Continued)

Pseudocode (Data-type definition)

```
struct Edge {
    HalfEdge e[2]; // pair of symmetric half-edges
};

struct HalfEdge {
    int index;      // index of half-edge in parent edge
    HalfEdge* next; // next CCW half-edge around left face
    Vertex* term;   // terminal vertex
    Face* left;     // left face
};
```

Example (Pictorial view of data structure)



Quad-Edge Data Structure

- proposed in:
  L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, Apr. 1985.
- simultaneously represents graph and its dual
- each edge belongs to four circular singly-linked lists corresponding to two vertices and two faces incident to edge
- vertex/face represented by ring of quad-edges
- 8 pointers + 4 two-bit integers per edge
- used in various research software available on Internet (e.g., Scape terrain-simplification software, Dani Lischinski's constrained DT software)

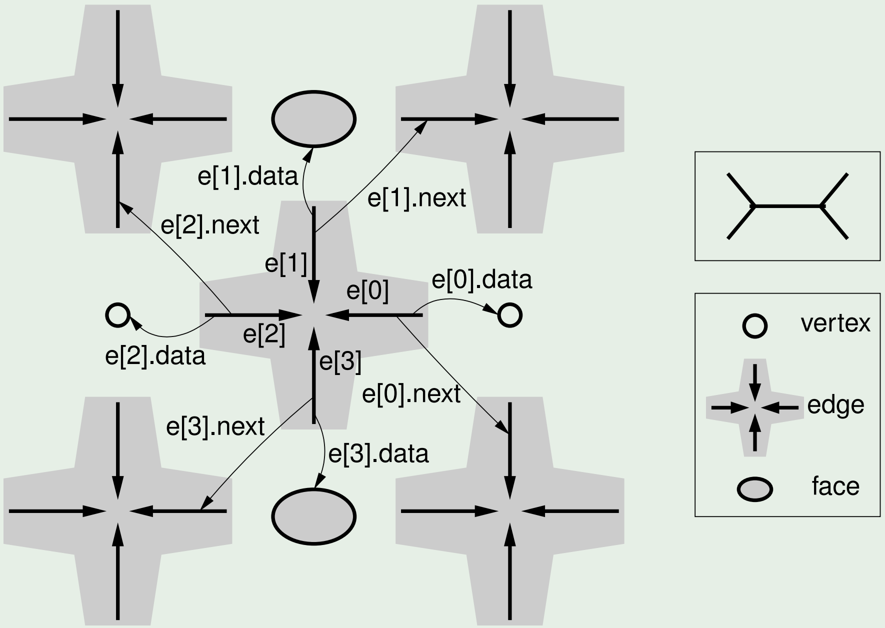
Quad-Edge Data Structure (Continued)

Pseudocode (Data-type definition)

```
struct Edge {
    QuadEdge* e[4]; // four quad-edges of edge
};

struct QuadEdge {
    int index;      // index of quad-edge in parent edge
    QuadEdge* next; // next CCW quad-edge with same origin
    void* data;     // face or vertex
}
```

Example (Pictorial view of data structure)



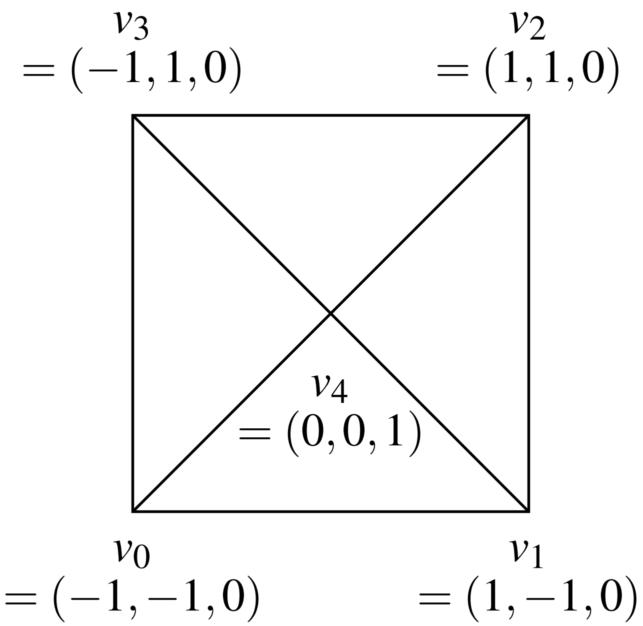
Comments on MATLAB

- MATLAB is well suited to numerical applications that require only simple data types such as arrays (e.g., vectors and matrices).
- MATLAB is not suitable for any practical computation involving polygon meshes.
- The data structures necessary for allowing efficient processing of mesh data are significantly more complex than the simple array types provided in MATLAB.

Object File Format (OFF)

- simple scheme for encoding the geometry and topology of a polygon mesh
- also has provisions for including color and normal information

OFF Example (Triangle Mesh)

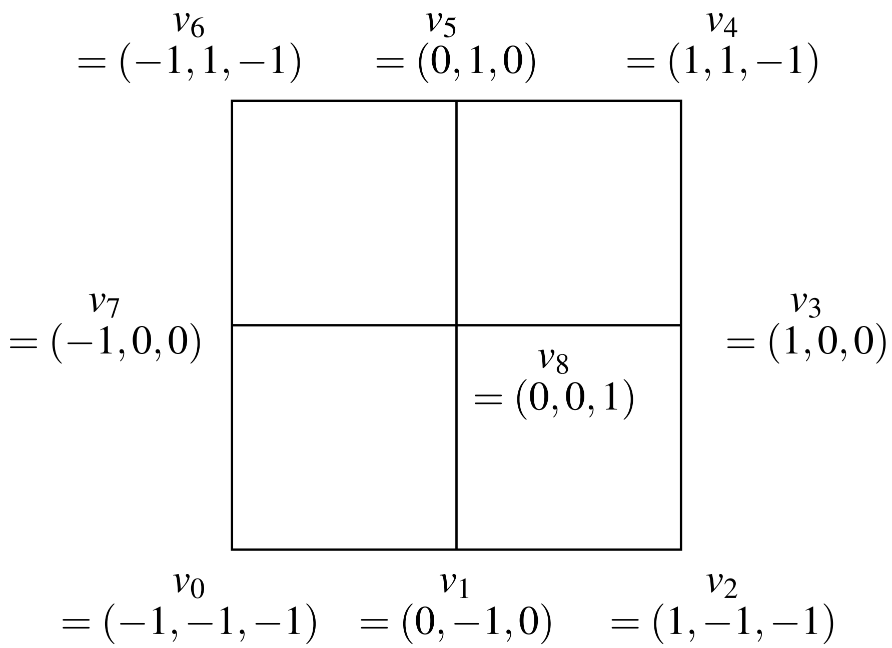


Mesh

```
OFF
5  4  0
-1  -1  0
 1  -1  0
 1   1  0
-1   1  0
 0   0  1
3  0  1  4
3  1  2  4
3  2  3  4
3  0  4  3
```

Corresponding OFF File

OFF Example (Quad Mesh)



Mesh

```
OFF
9 4 0
-1 -1 -1
 0 -1  0
 1 -1 -1
 1  0  0
 1  1 -1
 0  1  0
-1  1 -1
-1  0  0
 0  0  1
4 0 1 8 7
4 1 2 3 8
4 8 3 4 5
4 7 8 5 6
```

Corresponding OFF File

Section 5.6

Software and Datasets

MeshLab

- allows editing of polygon meshes
- supports Loop and Butterfly subdivision
- can do mesh simplification
- can save rendered output to image file in various formats
- available for Microsoft Windows and Unix/Linux platforms
- some Linux distributions have package for MeshLab (e.g., Fedora package: `meshlab`)
- home page: `http://meshlab.sourceforge.net`

Geomview

- interactive 3D mesh viewing program
- can produce rendered output in PostScript format
- fairly flexible, but user interface is not very intuitive
- supports Microsoft Windows and Unix/Linux platforms
- some Linux distributions have package for Geomview (e.g., Fedora package: `geomview`)
- home page: `http://www.geomview.org`

Blender

- 3D content creation suite
- supports modelling, shading, animation, rendering, imaging and compositing, real-time 3D/game creation
- implements Catmull-Clark subdivision surfaces for modelling
- available for Microsoft Windows and Unix/Linux platforms
- extremely powerful software, but might take some time to learn well
- some Linux distributions have package for Blender (e.g., Fedora package: `blender`)
- home page: `http://www.blender.org`

3-D Models

- Stanford 3-D Scanning Repository:
  `http://graphics.stanford.edu/data/3Dscanrep`
- NASA 3-D Resources:
  `http://www.nasa.gov/multimedia/3d_resources/models.html`
- CNRS Mesh Watermarking Datasets:
  `http://liris.cnrs.fr/meshbenchmark`
- Princeton Suggestive Contour Datasets:
  `http://www.cs.princeton.edu/gfx/proj/sugcon/models`

Part 6

Subdivision Surfaces and Subdivision Wavelets

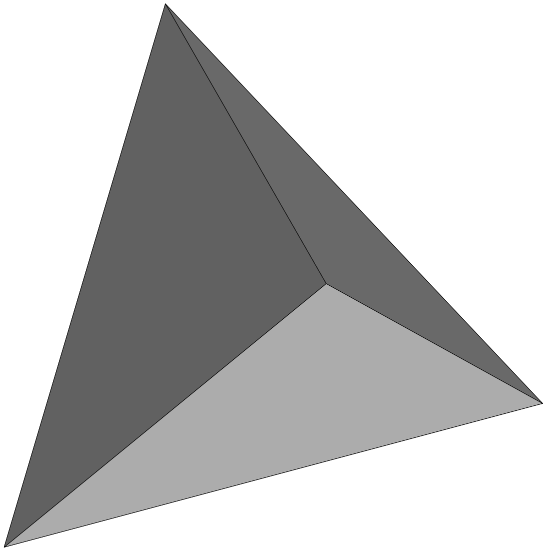
Section 6.1

Subdivision Surfaces

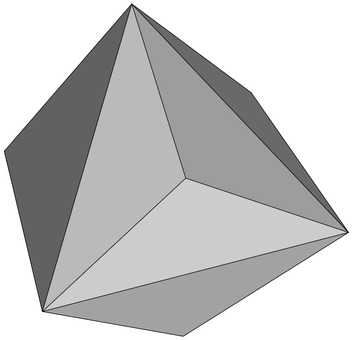
Subdivision Surfaces

- In many applications, polygon meshes are used to model surfaces.
- Modelling smooth surfaces with polygon meshes is problematic.
- To obtain a reasonably good approximation to a smooth surface with a polygon mesh, a very fine mesh with an extremely large number of faces is typically required.
- Subdivision solves this problem by representing smooth surfaces in terms of a ***coarse*** mesh.
- Subdivision provides a set of well-defined rules for producing successively refined versions of a mesh.
- Typically, these rules are chosen so that repeating the refinement process ad infinitum produces, in the limit, a smooth (or mostly smooth) surface.

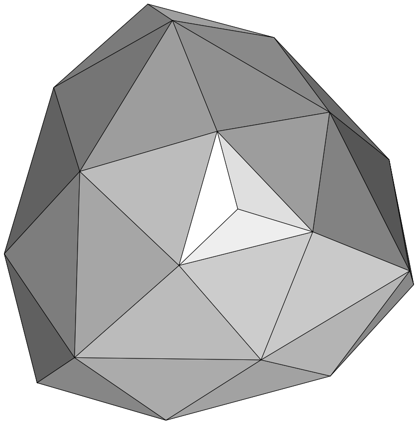
Subdivision Surface Example



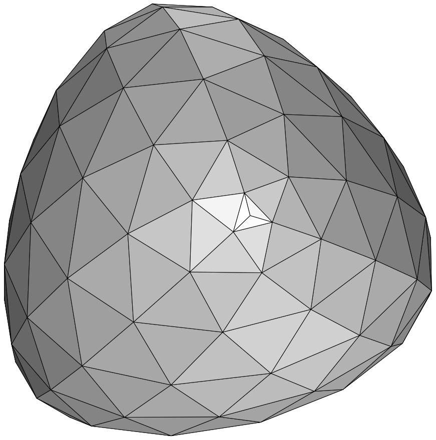
Control Mesh



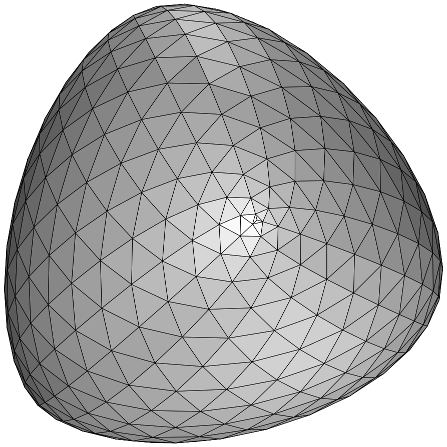
Mesh After One Iteration of Subdivision



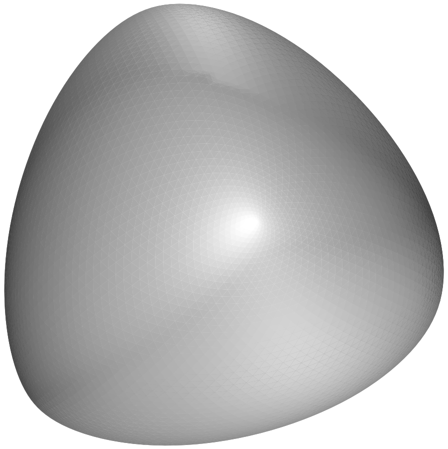
Mesh After Two Iterations of Subdivision



Mesh After Three Iterations of Subdivision



Mesh After Four Iterations of Subdivision



Limit Surface

Subdivision

- Subdivision is a general method for constructing a finer mesh from a coarser one, through the introduction of new vertices (as well as edges and faces).
- The coarser mesh that serves as the starting point for subdivision is called a **control mesh** (or base mesh).
- The addition of new vertices to a mesh requires modifications to both the topology (i.e., connectivity) and geometry (i.e., vertex positions) of the mesh.
- For this reason, each subdivision scheme requires the specification of two rules:
  1. a **topologic refinement rule** that describes how the connectivity of the mesh is to be modified in order to incorporate the new vertices being added to the mesh; and
  2. a **geometric refinement rule** that describes how the geometry of the mesh is to be changed in order to accommodate the new vertices being added (where these modifications may affect the position of previously-existing vertices).

Subdivision (Continued 1)

- In practice, the refinement rules are applied repeatedly until a mesh of the desired fineness is obtained.
- Provided that a subdivision scheme is well behaved, if the refinement rules are applied repeatedly ad infinitum, the vertices in the mesh will converge, in the limit, to a surface. Such a surface is called a **limit surface**.
- A subdivision scheme is said to be **interpolating** if it always produces refined meshes that pass through all of the vertices of the original control mesh. Otherwise, the scheme is said to be **approximating**.
- The (topologic and geometric) refinement rules employed in subdivision are chosen in order to obtain refined meshes with certain desirable properties.

Subdivision (Continued 2)

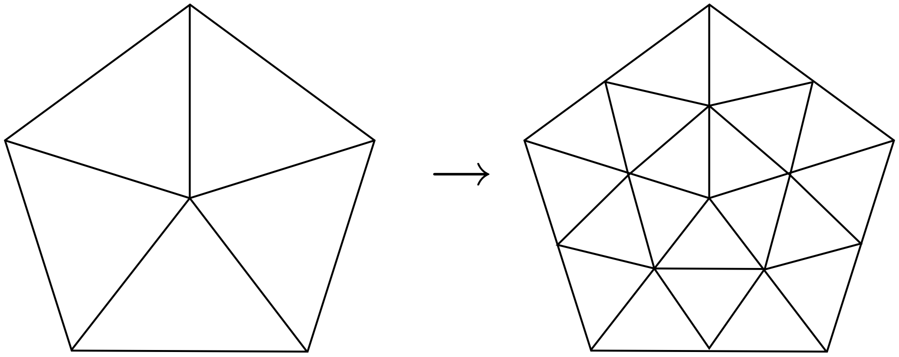
- Perhaps, most importantly, the refinement rules are chosen to ensure that subdivision will always converge to a limit surface. Without a guarantee of convergence, the subdivision process would likely produce very poorly-behaved meshes.
- In many applications, smooth meshes are highly desirable. Consequently, the refinement rules are usually chosen to ensure that the limit surface be as smooth as possible.
- For example, in some computer graphics applications, at least C^2 continuity is desirable.
- If the same topologic and geometric rules are used in each iteration of a subdivision scheme, the scheme is said to be **stationary**.
- If the same geometric rule is used to calculate the vertices within a single iteration of a subdivision scheme, the scheme is said to be **uniform**.

Topologic Refinement Rule

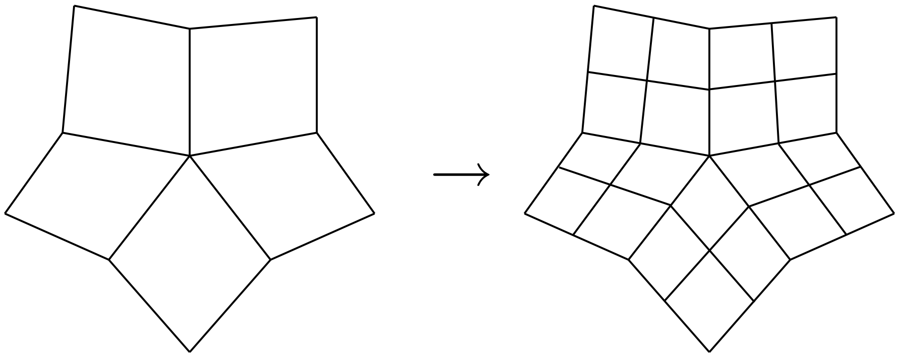
- The topologic refinement rule specifies how the ***connectivity*** of the mesh is to be modified in order to incorporate new vertices into the mesh. [***Note:*** Such a rule does not say anything about the coordinates of vertices, as this constitutes geometric information.]
- Generally, there are two types of topologic rules: primal and dual. A primal scheme splits faces. A dual scheme splits vertices.
- Since a topologic rule in subdivision introduces new vertices such that they are connected in a regular (i.e., highly structured) fashion, new vertices will always have particular valences.
- New vertices introduced in the interior will all have the same valence.
- In the case of a triangle mesh, new vertices have a valence of six in the interior.
- In the case of a quadrilateral mesh, new vertices have a valence of four in the interior.
- A vertex with one of these special valence values is said to be **regular**.
- A vertex that is not regular is said to be **extraordinary**.

Examples of Topologic Refinement Rules

- Two common topologic refinement rules are shown below pictorially.



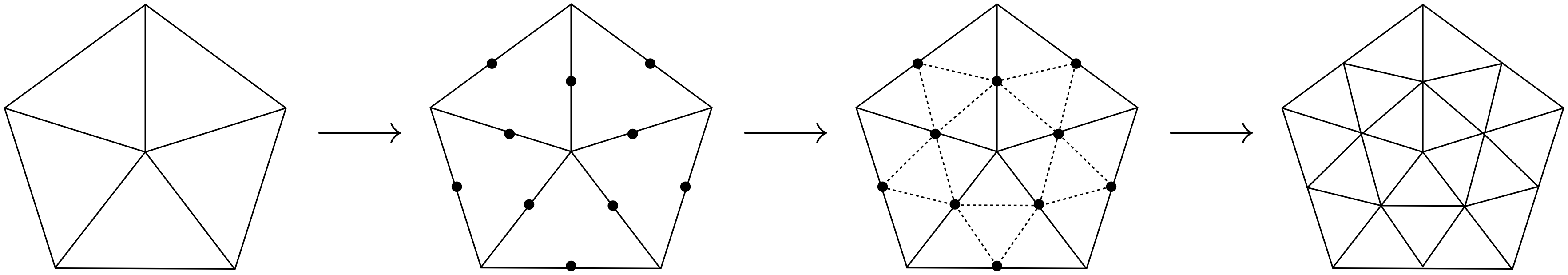
Primal triangle quadrisection



Primal quadrilateral quadrisection

Primal Triangle Quadrisection

- Each edge in the original mesh is split in two, with a new vertex being inserted at the location of the split. Each of these newly added vertices is referred to as an **edge vertex**.
- Each new vertex is then connected by an edge to each of the other new vertices that originated from the same face (before edge splitting).
- The preceding process is illustrated below.



- Observe that all new vertices added in the interior have valence six, while all new vertices added on the boundary have valence four.
- Note that no mention is made as to what the coordinates of the new vertices are. They are essentially undefined. A geometric refinement rule is needed to specify how to assign coordinates to the new vertices.

Geometric Refinement Rule

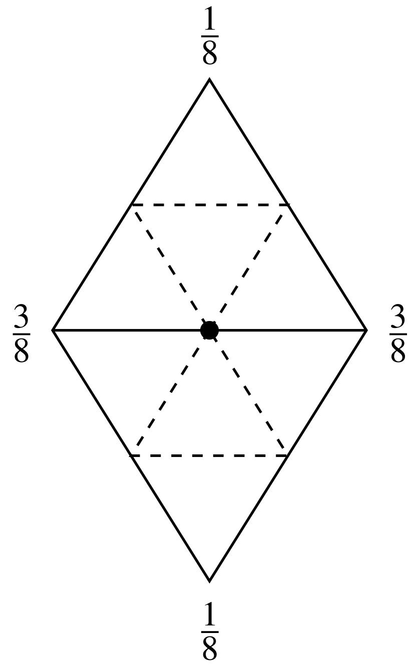
- The geometric refinement rule specifies how the geometry of the mesh (i.e., the position of vertices) is to be changed in order to accommodate the new vertices being added.
- Clearly, the rule must specify how to determine the position of any new vertices introduced by the topologic refinement rule.
- In addition, the rule must indicate how to handle the old (i.e., previously existing) vertices, which may be either modified or left unchanged.
- If the rule leaves the old vertex positions unchanged, this leads to an interpolating scheme.
- The geometric refinement rule can be viewed as a filtering operation for meshes.

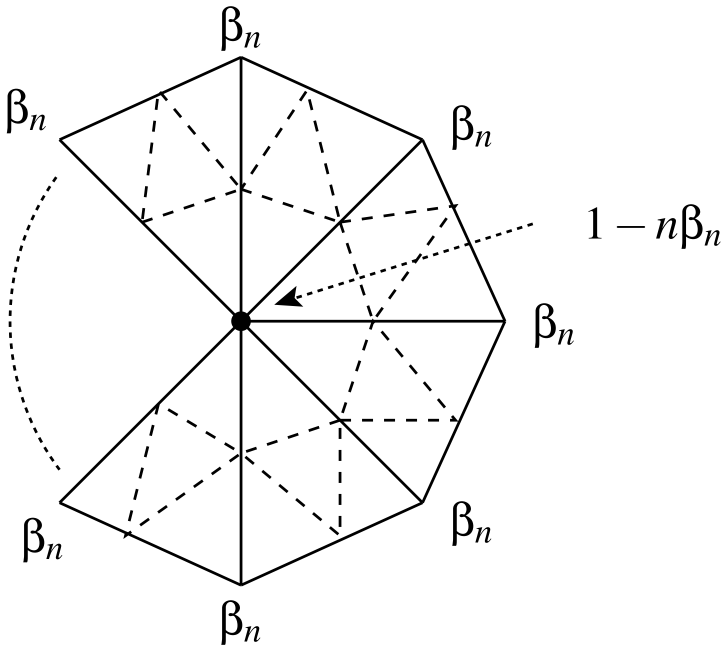
Geometric Refinement Rule (Continued)

- The mask coefficients normally sum to one in order to correspond to affine combinations of points (and achieve affine invariance).
- Normally, in the case of meshes with boundary, the masks used to determine the positions of boundary vertices depend only on boundary vertices.
- The rule is specified in terms of masks, which are often expressed pictorially.

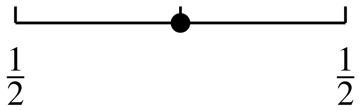
Examples of Geometric Refinement Rules

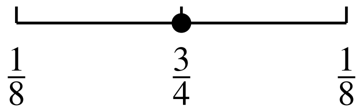
- Some examples of masks are shown below, expressed in pictorial form.





$$\beta_n = \frac{1}{n}\left[\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n}\right)^2\right]$$





Classification of Subdivision Schemes

- Subdivision schemes can be classified using a variety of criteria:
  1. the type of mesh can be handled by the scheme (e.g., triangle, quadrilateral, triangle/quadrilateral, hexagonal);
  2. whether the scheme is approximating or interpolating;
  3. whether the scheme is primal (i.e., based on face splitting) or dual (i.e., based on vertex splitting);
  4. the smoothness of the limit surface produced by the scheme (e.g., C^0, C^1, or C^2 continuity).

Subdivision Schemes

Scheme	Attributes
Linear	triangle; interpolating; C^0
Bilinear	quadrilateral; interpolating; C^0
Midedge	quadrilateral; approximating; dual; C^1
Doo-Sabin	quadrilateral; approximating; dual; C^1
Catmull-Clark	polygon/quadrilateral; approximating; primal; C^2 everywhere except at extraordinary points where C^1
Loop	triangle; approximating; primal; C^2 everywhere except at extraordinary points where C^1
Butterfly	triangle; interpolating; primal; C^1 continuous everywhere except at extraordinary points
Modified Butterfly	triangle; interpolating; primal; C^1 continuous everywhere
Kobbelt $\sqrt{3}$	triangle; approximating; primal; C^2 everywhere except at extraordinary points where C^1
Stam-Loop	quadrilateral/triangle; approximating; primal?

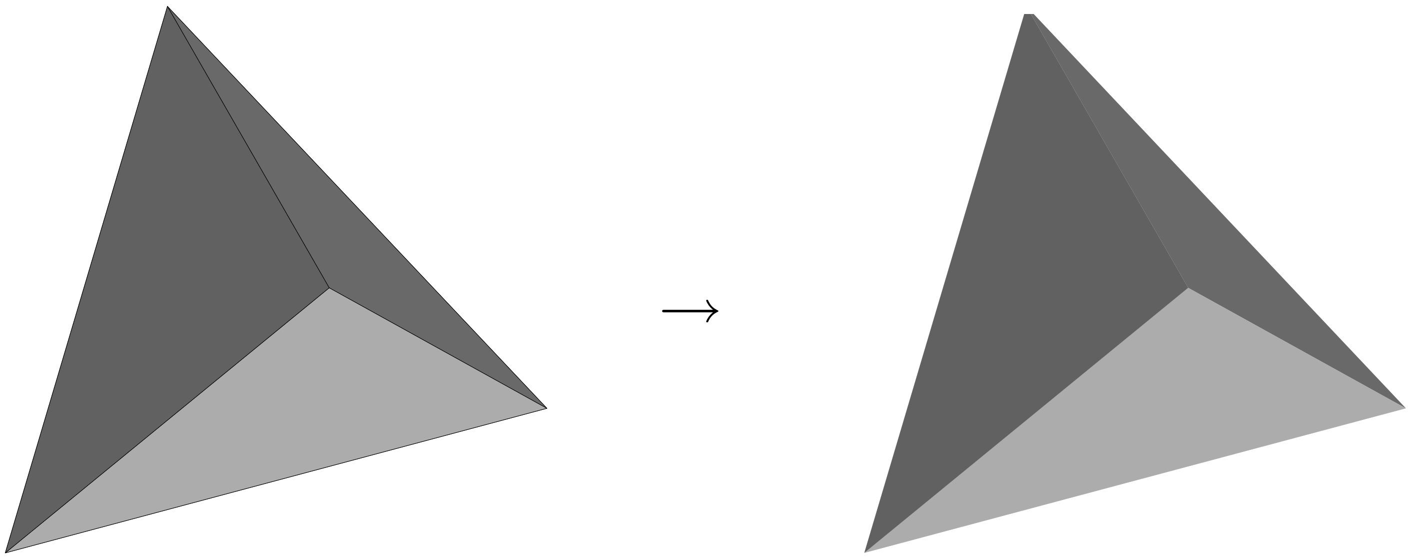
Section 6.1.1

Specific Examples of Subdivision Schemes

Linear Subdivision

- This subdivision scheme is defined for ***triangle meshes***. It is ***primal*** and ***interpolating***.
- This scheme is extremely simple.
- With this scheme, the limit surface is always identical to the original control mesh. Therefore, this surface is only guaranteed to be C^0 ***continuous***.
- The topologic refinement rule employed is ***primal triangle quadrisection***.
- The geometric rule places each new edge vertex at the midpoint of the edge (in the unrefined mesh) from which the new vertex was generated. That is, if v denotes a new edge vertex associated with an edge in the unrefined mesh having vertices v_1 and v_2, we choose $v = \frac{1}{2}(v_1 + v_2)$.
- This scheme is not particularly interesting, since the limit surface is simply equal to the original control mesh.

Linear Subdivision: Example



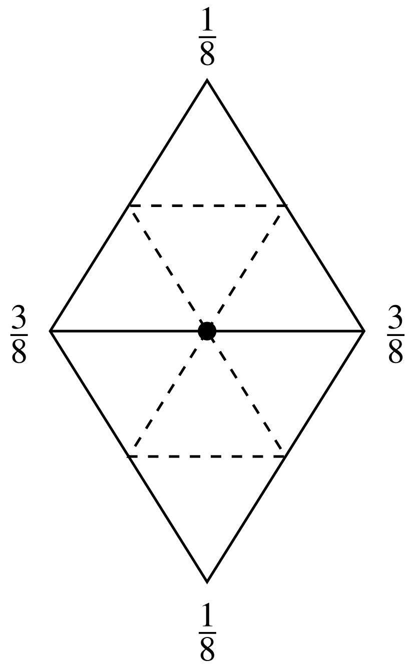
Loop Subdivision

- The Loop subdivision scheme was originally proposed in:

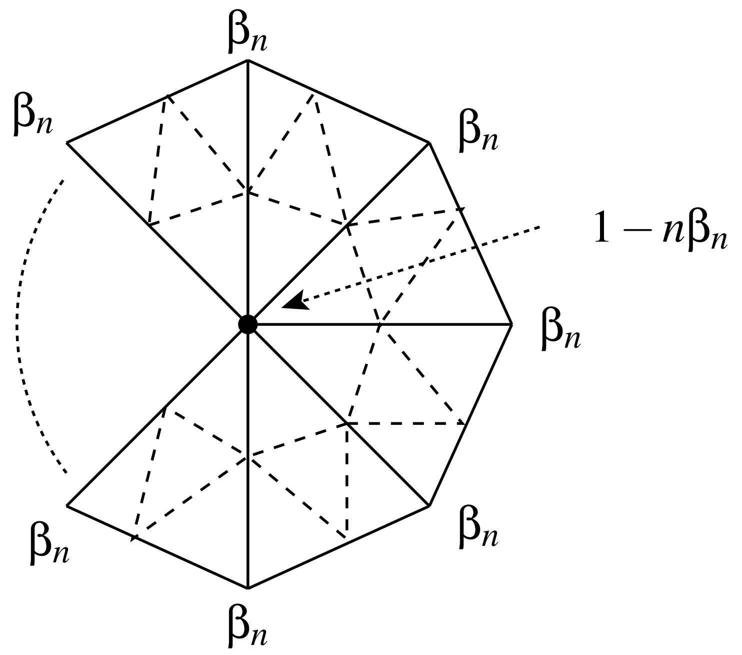
  C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, Department of Mathematics, University of Utah, 1987.

- This scheme is defined for ***triangle meshes***. It is ***approximating*** and ***primal***.
- This scheme produces limit surfaces that are C^2 ***continuous*** everywhere, except at extraordinary vertices where C^1 continuity is achieved.
- The topologic refinement rule employed is ***primal triangle quadrisection***.

Loop Subdivision: Geometric Refinement

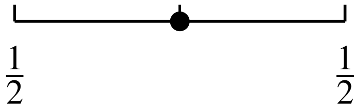


Mask for *edge* vertices
(*nonboundary*, regular)

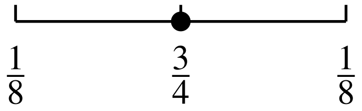


$$\beta_n = \frac{1}{n}\left[\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n}\right)^2\right]$$

Mask for *non-edge* vertices of valence n
(*nonboundary*, regular, and extraordinary)



Mask for *edge* vertices
(*boundary*)



Mask for *non-edge* vertices
(*boundary*)

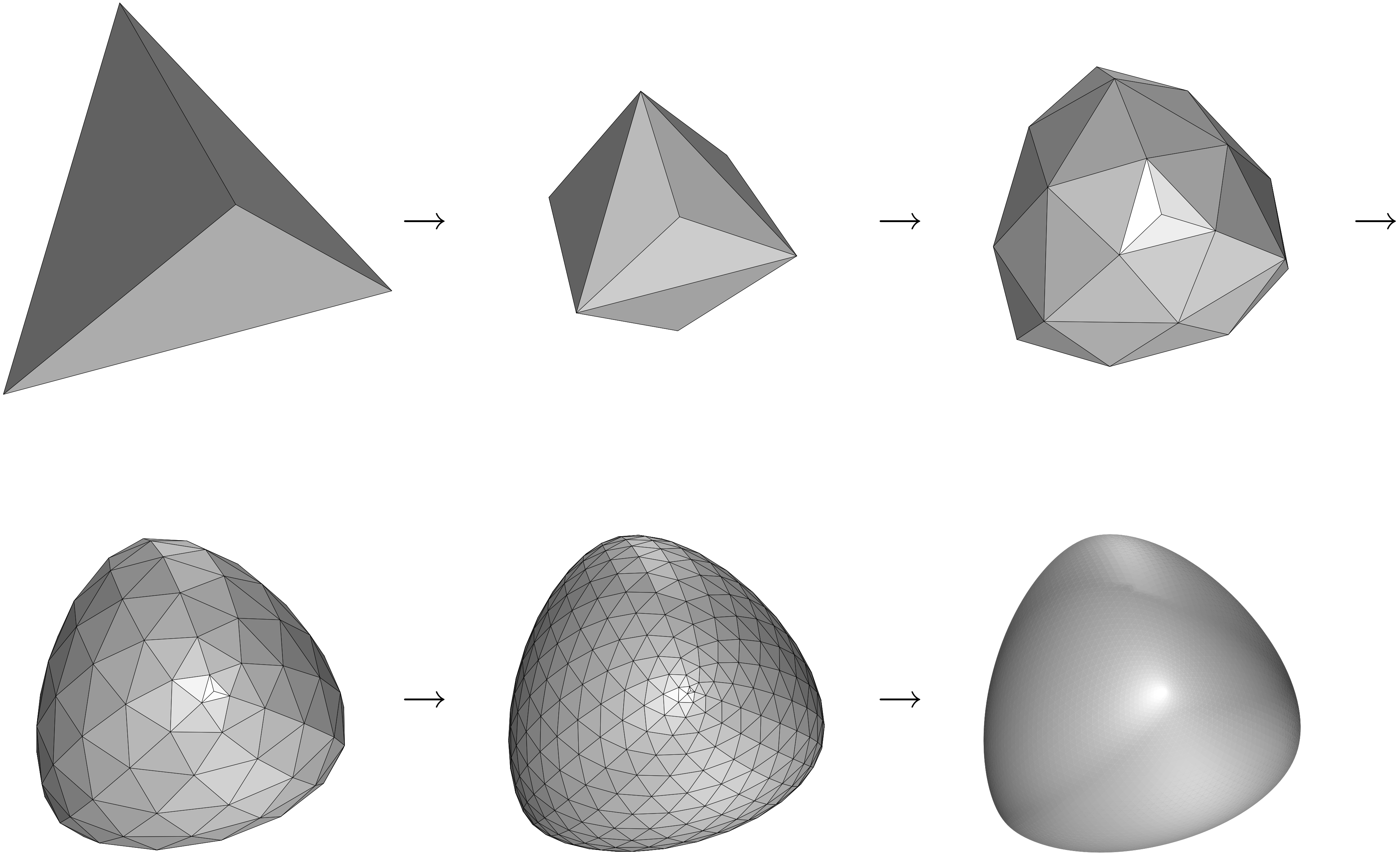
Loop Subdivision: Variations on Loop Subdivision

- Sometimes we may want to introduce creases in the smooth limit surface resulting from subdivision. In this case, we treat the location of a crease as a boundary and apply the appropriate boundary rule.
- A modified version of Loop subdivision proposed by Warren and Weimer chooses β_n as

$$\beta_n = \begin{cases} \frac{3}{8n} & n > 3 \\ \frac{3}{16} & n = 3. \end{cases}$$

The resulting coefficients have “nicer” values than those in the original Loop scheme.

Loop Subdivision: Example



Butterfly Subdivision

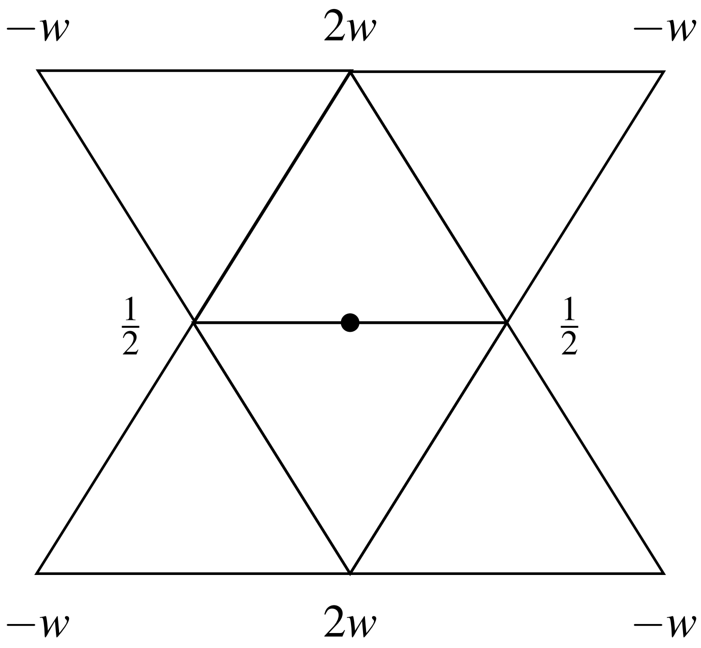
- The butterfly subdivision scheme was originally proposed in:

  N. Dyn, D. Levin, and J. A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9(2):160–169, 1990.

- This subdivision scheme is defined for ***triangle meshes***. It is ***interpolating*** and ***primal***.
- The limit surfaces produced by this scheme are C^1 ***continuous*** everywhere except at extraordinary vertices of valence k where $k = 3$ or $k > 7$. At these points, the surface is only C^0 continuous.
- The topologic refinement rule employed is ***primal triangle quadrisection***.
- A tension parameter allows local control over smoothness. For example, it can be used to form creases in the limit surface.

Butterfly Subdivision: Geometric Refinement

- The boundary cases are somewhat complicated and ignored here.



tension parameter w nominally chosen as

$$w = \frac{1}{16}$$

Mask for edge vertices
(regular)

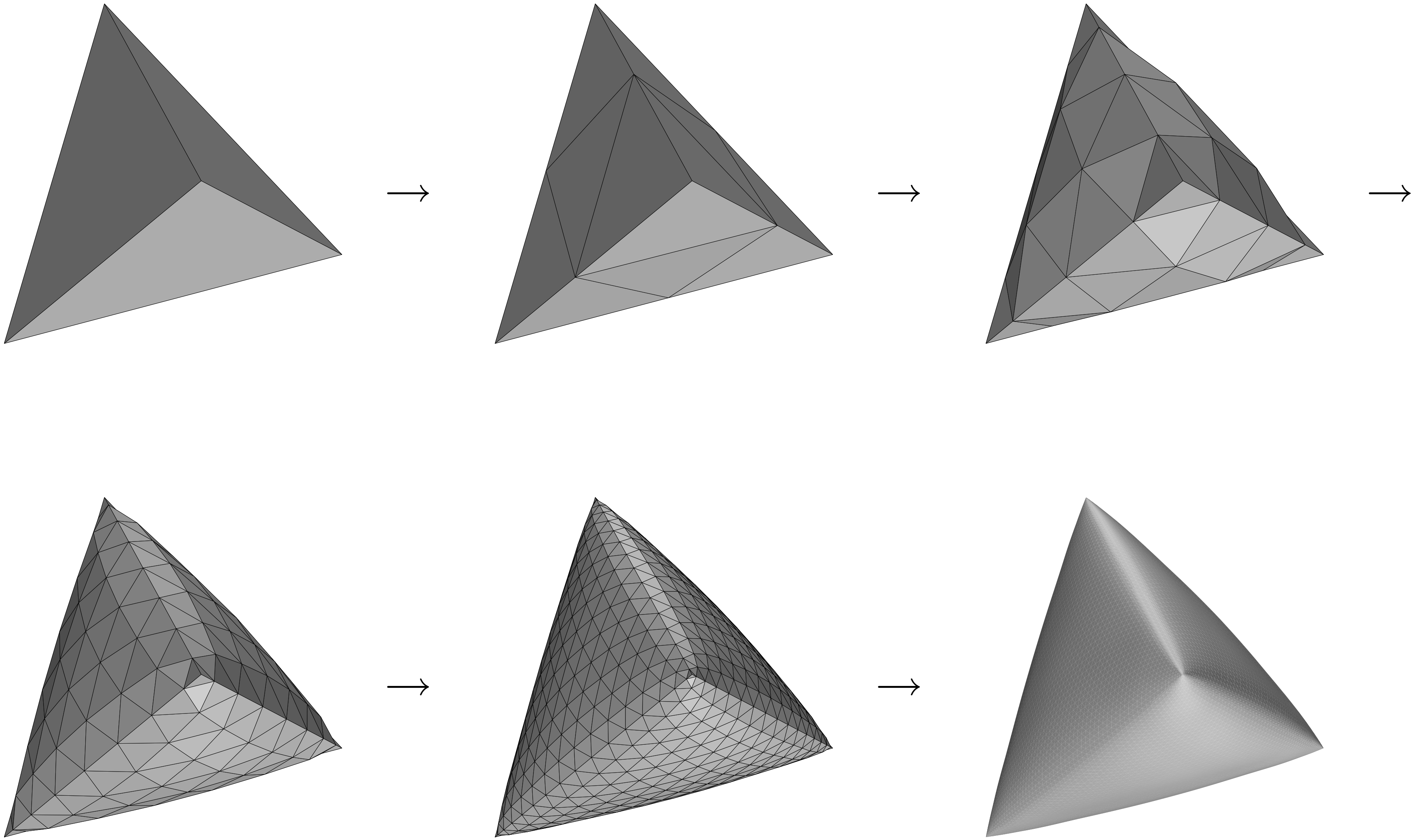
Butterfly Subdivision: Variants of Butterfly Subdivision

- A variant of the butterfly subdivision scheme called the modified butterfly scheme has been proposed in:

  > D. Zorin, P. Schroder, and W. Sweldens. Interpolating subdivision for meshes with arbitrary topology. In *SIGGRAPH 96*, pages 189–192, 1996.

- The limit surfaces produced are C^1 continuous everywhere (including all extraordinary points).

Butterfly Subdivision: Example



Kobbelt $\sqrt{3}$ Subdivision

- The Kobbelt $\sqrt{3}$ subdivision scheme was originally proposed in

  L. Kobbelt. $\sqrt{3}$ subdivision. In *Proc. of SIGGRAPH 2000*, pages 103–112, 2000.

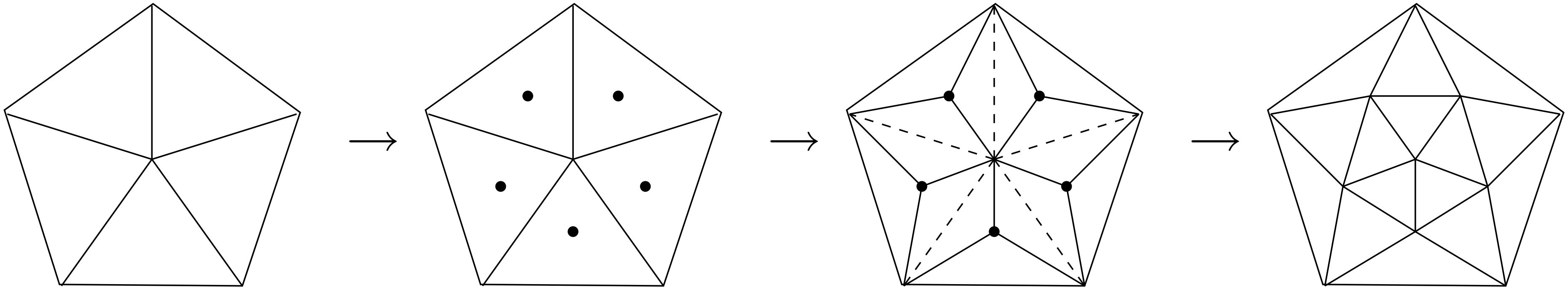
- The scheme is defined for ***triangle meshes***. It is ***primal*** and ***approximating***.
- The limit surfaces produced by this method are C^2 ***continuous*** everywhere, except at extraordinary points where C^1 continuity is obtained.
- The topologic refinement rule employed is sometimes referred to by the name $\sqrt{3}$, and will be discussed in more detail shortly.
- This subdivision scheme is better suited to adaptive refinement strategies (since it easily allows for local refinement without cracks).
- The number of vertices/faces increase more slowly with this subdivision scheme than with those employing primal triangle quadrisection.

$\sqrt{3}$ Topologic Refinement

- For triangle meshes, a more exotic topologic refinement rule is $\sqrt{3}$ refinement.
- This topologic refinement scheme employs two slightly different rules, where one is used in even iterations and one in odd iterations.
- These two cases only differ in how they handle boundaries.
- So, in the case of a mesh without a boundary, there is no difference in the processing of even and odd iterations.
- Iterations number from zero so that the first iteration is an even iteration.

$\sqrt{3}$ Topologic Refinement (Even Iteration)

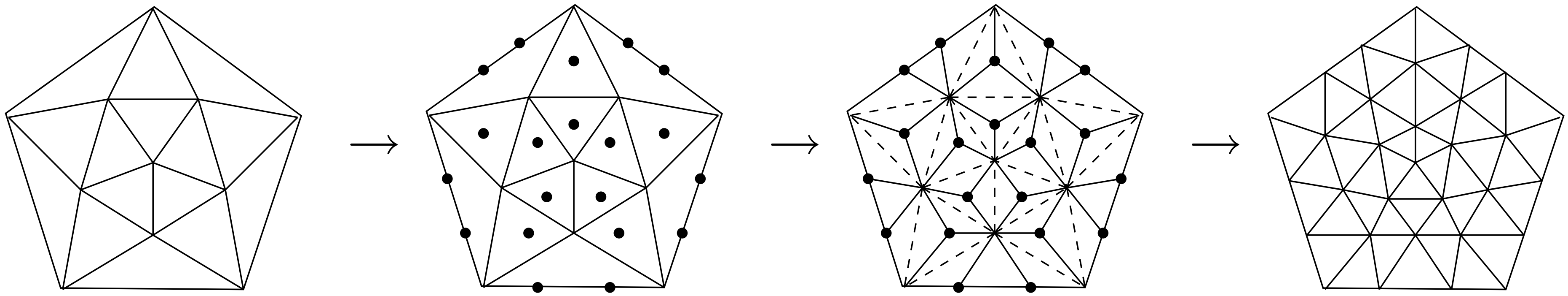
- In the interior of each face, a new vertex is added.
- Then, each new vertex is connected by edges to each of its three surrounding old vertices.
- Lastly, every original edge that connects two old vertices is flipped, except for boundary edges (which cannot be flipped).
- This process is illustrated pictorially below.



- All new vertices in the interior that are not incident on a boundary face have valence six.
- New vertices in the interior that are incident on a boundary face have valence between three and five.

$\sqrt{3}$ Topologic Refinement (Odd Iteration)

- Odd iterations are handled in a similar way as even iterations, except for how boundary edges are handled at the beginning of the refinement step.
- Due to the manner in which even iterations are handled, in an odd iteration, a triangle can have at most one edge on the boundary.
- In odd iterations, for each triangle with an edge on the boundary, the boundary edge is split twice inserting two new vertices along the edge.
- Then, the two new vertices are connected to the old vertex on the side of the triangle opposite the edge on which the two new vertices were inserted.
- The remainder of the refinement process is carried out identically to the even iteration case.
- This process is illustrated pictorially below.

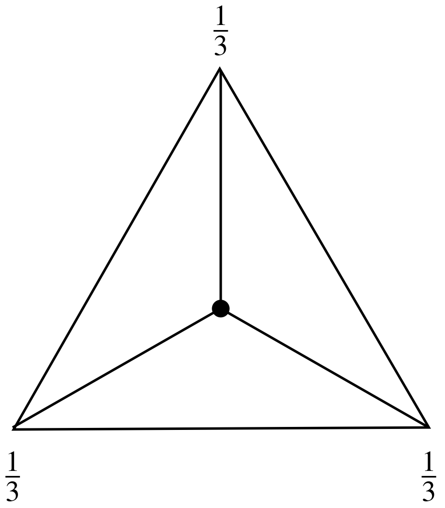


$\sqrt{3}$ Topologic Refinement

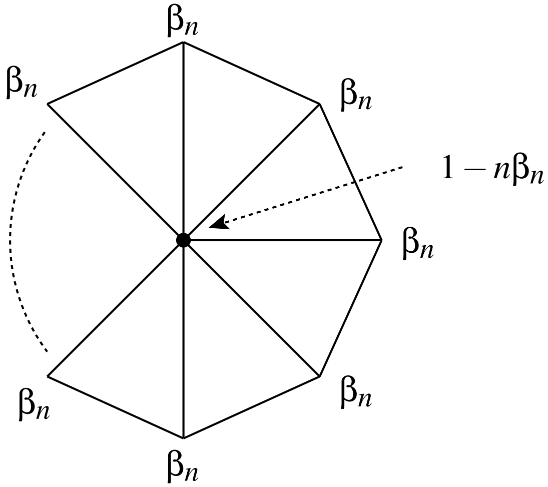
- Every two iterations of the topologic refinement process result in each original triangle being replaced by nine new triangles.
- All new vertices in the interior that are not incident on a boundary face have valence six.
- Applying topologic refinement twice results in a uniform refinement with trisection of every original edge.
- Each triangle is split into nine new triangles.
- A single iteration can be viewed as the “square root” of a three-fold split of each edge, hence the name $\sqrt{3}$.

Kobbelt $\sqrt{3}$ Subdivision: Geometric Refinement

- **New nonboundary vertex case:** Let p be the new vertex inserted into the face of the unrefined mesh having vertices p_1, p_2, p_3. Then, we choose $p = \frac{1}{3}(p_1 + p_2 + p_3)$ (i.e., p is inserted at the barycenter of its associated face).
- **Old nonboundary vertex case:** Let p be the old vertex to be updated, and let $p_1, p_2, \ldots, p_n$ be the (1-ring) neighbours of q in the unrefined mesh. Then, we update p to be $p' = (1 - n\beta_n)p + \beta_n \sum_{k=1}^{n} p_k$, where $\beta_n = \frac{1}{9n}\left[4 - 2\cos\left(\frac{2\pi}{n}\right)\right]$.



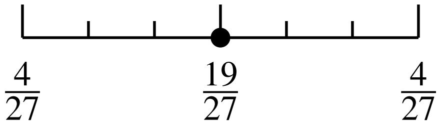
Mask for ***new nonboundary*** vertex



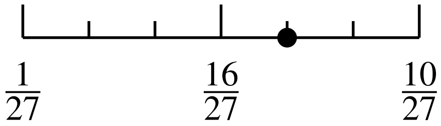
Mask for ***old nonboundary*** vertex

Kobbelt $\sqrt{3}$ Subdivision: Geometric Refinement (Continued)

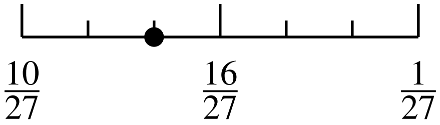
- In even iterations, the old boundary vertices are left unchanged.
- In odd iterations, boundary vertices are handled as shown below.



Mask for ***old boundary*** vertex (***odd iteration***)

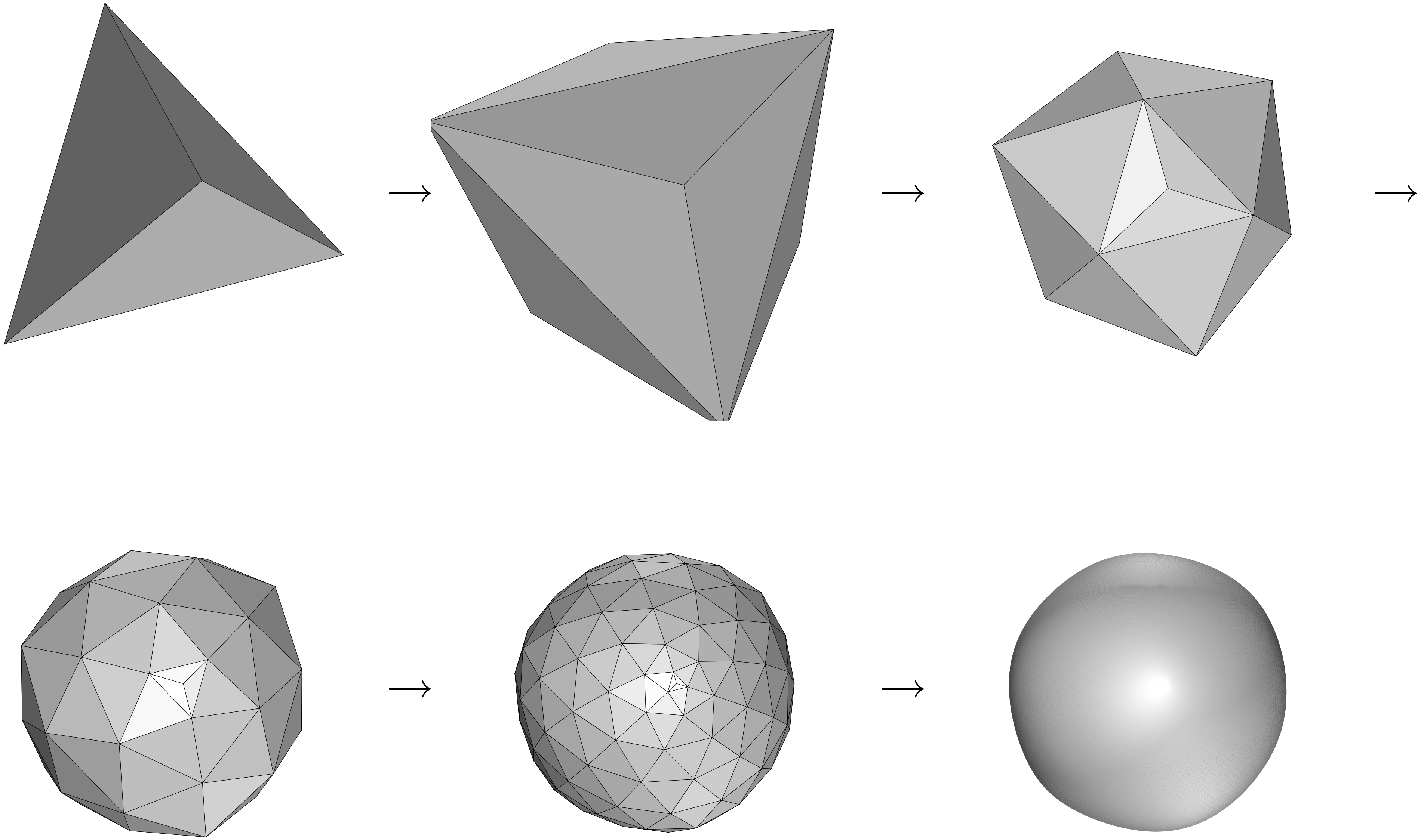


Mask for ***new boundary*** vertex (***odd iteration***)



Mask for ***new boundary*** vertex (***odd iteration***)

Kobbelt $\sqrt{3}$ Subdivision: Example



Catmull-Clark Subdivision

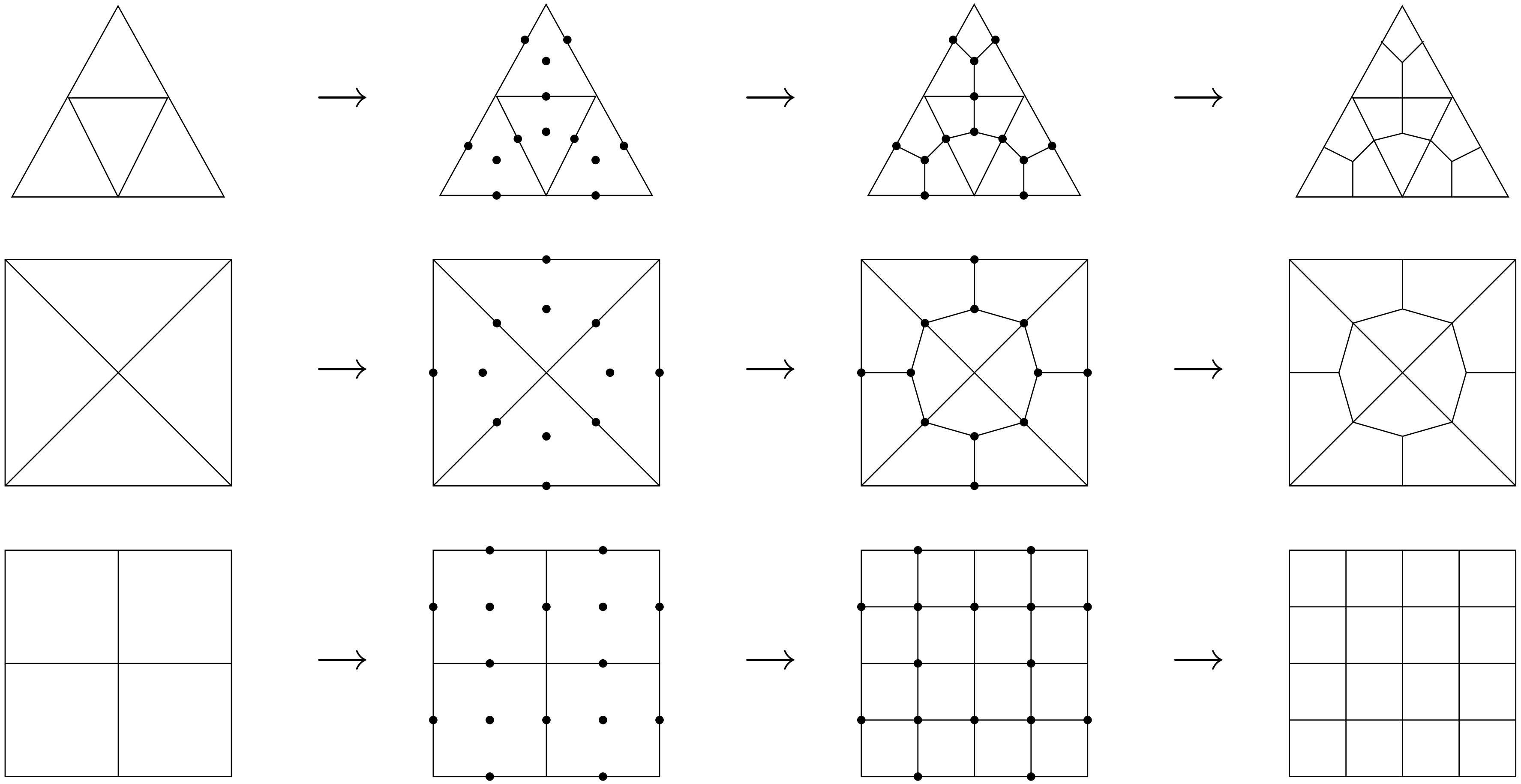
- The Catmull-Clark subdivision scheme was originally proposed in:

  E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6):350–355, 1978.
- Although Catmull-Clark subdivision was historically one of the first subdivision methods proposed, it is still used quite frequently in practice today, especially in computer animation.
- This method is a generalization of bicubic B-splines to arbitrary meshes. That is, in the case of a regular quadrilateral mesh, Catmull-Clark subdivision yields a bicubic B-spline surface.
- This scheme can be applied to polygon meshes with ***any type of faces*** (e.g., triangle, quadrilateral, triangle/quadrilateral, hexagonal).
- The scheme is ***primal*** (i.e., face splitting) and approximating.
- The limit surfaces produced have C^2 ***continuity*** everywhere, except at extraordinary points where only C^1 continuity is achieved.
- The subdivision process always produces quadrilateral faces, regardless of the types of polygons in the control mesh.

Catmull-Clark Subdivision: Topologic Refinement

- A new vertex is inserted in each face. Such a vertex is called a **face point**.
- Each edge is split in two by inserting a new vertex on the edge. Such a vertex is called an **edge point**.
- Then, each face point is connected by an edge to each of the edge points associated with the same face.
- This topologic refinement rule effectively splits an n-gon into n quadrilaterals.
- For this reason, subdivision ***always produces quadrilateral meshes***.
- For a quadrilateral input mesh, this topologic refinement rule is simply equivalent to primal quadrilateral quadrisection.
- The first iteration of subdivision may introduce ***new extraordinary vertices***. All subsequent iterations, however, introduce ***only regular*** (i.e., valence four) vertices in the interior of the mesh.

Catmull-Clark Subdivision: Topologic Refinement (Continued)



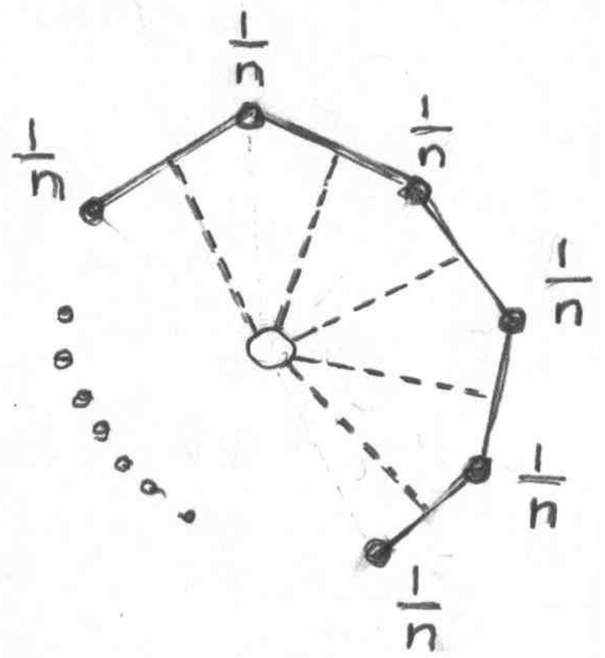
Catmull-Clark Subdivision: Geometric Refinement

- ***Face point:*** A face point v is chosen as the average of the vertices defining its associated face. That is, a face point v associated with the old face having vertices $v_0, v_1, \ldots, v_{n-1}$ is given by $v = \frac{1}{n}\sum_{k=0}^{n-1} v_k$.
- ***Edge point (nonboundary case):*** An edge point v is chosen as the average of the midpoint of the old edge and the average of the two new face points of the faces sharing the edge. Suppose that an edge point v originated from splitting an old edge that has vertices v_0, v_1 and is incident on the old faces associated with face points f_0, f_1. Then, $v = \frac{1}{2}\left[\frac{1}{2}(v_0 + v_1) + \frac{1}{2}(f_0 + f_1)\right] = \frac{1}{4}(v_0 + v_1 + f_0 + f_1)$.
- ***Edge point (boundary case):*** An edge point is chosen as the midpoint of its corresponding old edge. That is, for an edge point v originating from splitting an old edge with vertices v_0, v_1, we choose $v = \frac{1}{2}(v_0 + v_1)$.

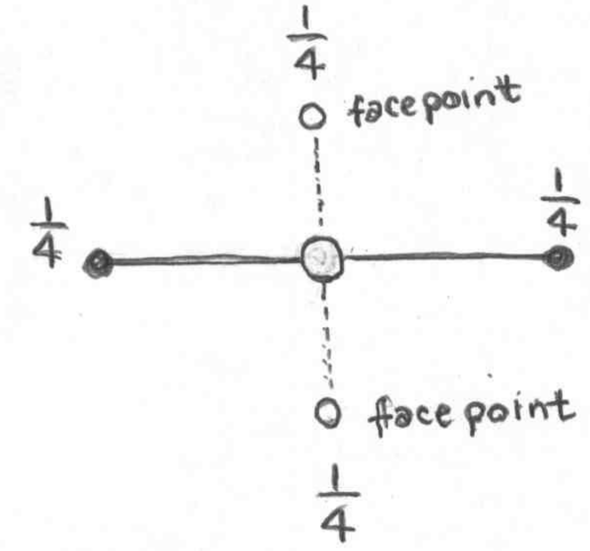
Catmull-Clark Subdivision: Geometric Refinement (Continued 1)

- ***Old vertex (nonboundary case):*** Consider an old vertex v with valence n in the old mesh. Let $v_0, v_1, \ldots, v_{n-1}$ be the old vertices that are 1-ring neighbours of v (before refinement). Let $f_0, f_1, \ldots, f_{n-1}$ be the face points of all faces incident on v. Then, the new value v' for the old vertex is given by $v' = \frac{n-3}{n}v + \frac{1}{n}q + \frac{2}{n}r$, where $q = \frac{1}{n}\sum_{k=0}^{n-1} f_k$, and $r = \frac{1}{n}\sum_{k=0}^{n-1}\frac{1}{2}(v + v_k)$ (i.e., q is the average of the new face points of all faces adjacent to the old vertex point, and r is the average of the midpoints of all old edges incident on the old vertex point). [Alternatively, one can show that $v' = \frac{n-2}{n}v + \frac{1}{n^2}\left(\sum_{k=0}^{n-1} v_k + \sum_{k=0}^{n-1} f_k\right)$.]
- ***Old vertex (boundary case):*** The old vertex v with the neighbouring boundary vertices v_0, v_1 has its new value v' chosen as $v' = \frac{3}{4}v + \frac{1}{8}(v_0 + v_1)$.

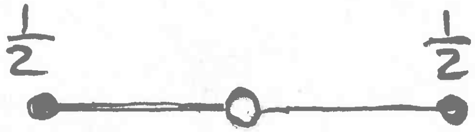
Catmull-Clark Subdivision: Geometric Refinement (Continued 2)



Face Point



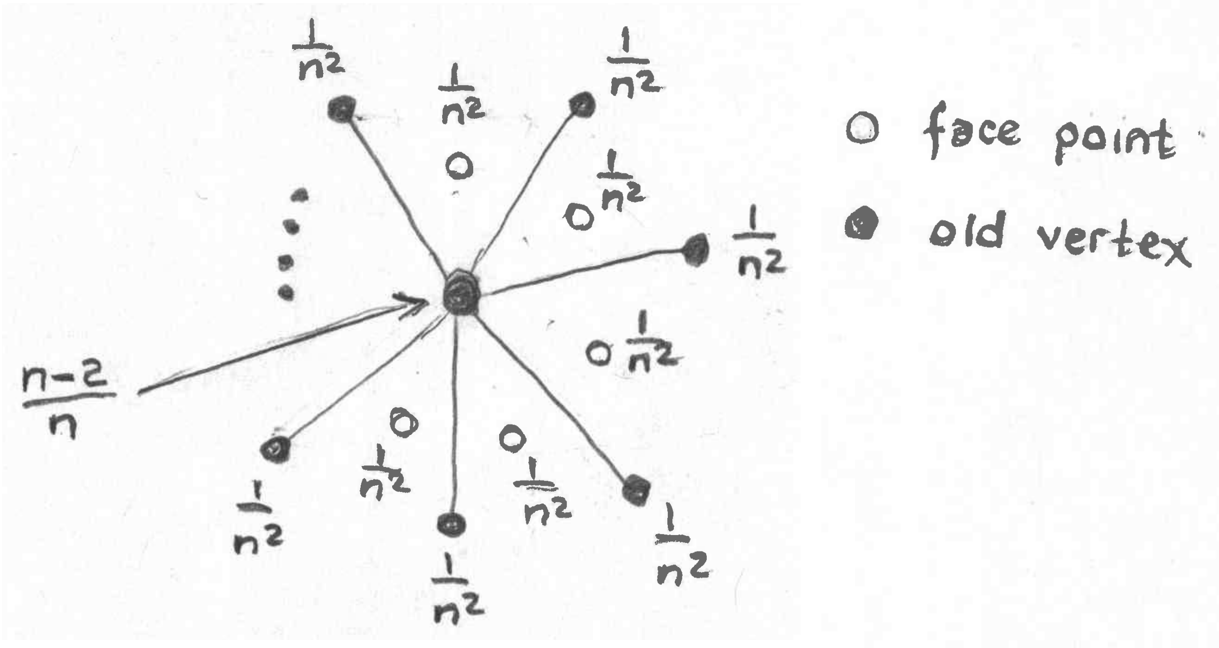
Nonboundary Edge Point



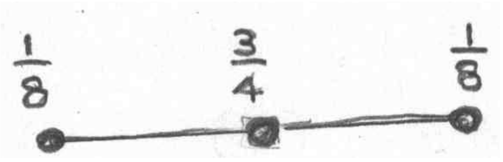
Boundary Edge Point

- Note: These diagrams have a somewhat different form from the geometric refinement masks presented elsewhere. (Open circles are new points added by subdivision.)

Catmull-Clark Subdivision: Geometric Refinement (Continued 3)



Old Nonboundary Vertex



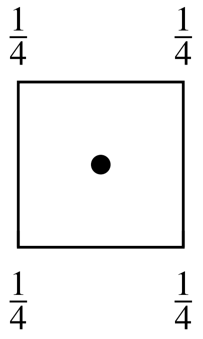
Old Boundary Vertex

- Note: These diagrams have a somewhat different form from the geometric refinement masks presented elsewhere. (Open circles are new points added by subdivision.)

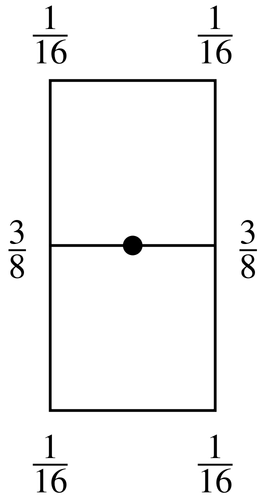
Catmull-Clark Subdivision: Geometric Refinement (Continued 4)

- Although all new faces introduced by subdivision are quadrilaterals, these faces are ***not necessarily planar***.
- In some situations, non-planar faces may be undesirable. For example, some rendering engines may not properly handle non-planar faces, which can lead to ***undesirable artifacts*** appearing in the rendered image.
- Since the control mesh can be a polygon mesh with any types of faces, it is not practical to enumerate all possible cases of geometric masks in pictorial form.
- In the case that the mesh to be refined is a ***quadrilateral mesh*** (which is always the case after the first iteration of subdivision), the masks have the form shown on the ***next slide***.

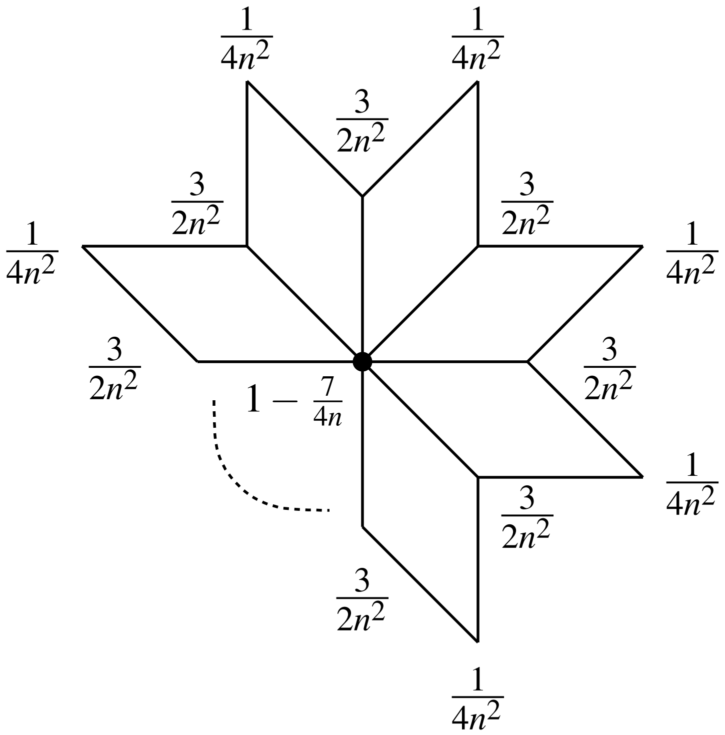
Catmull-Clark Subdivision: Geometric Refinement Masks



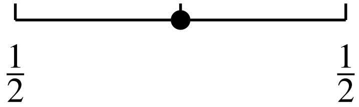
Face Point



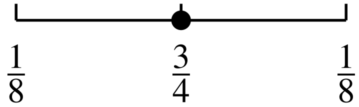
Edge Point (Interior)



Old Vertex (Interior)

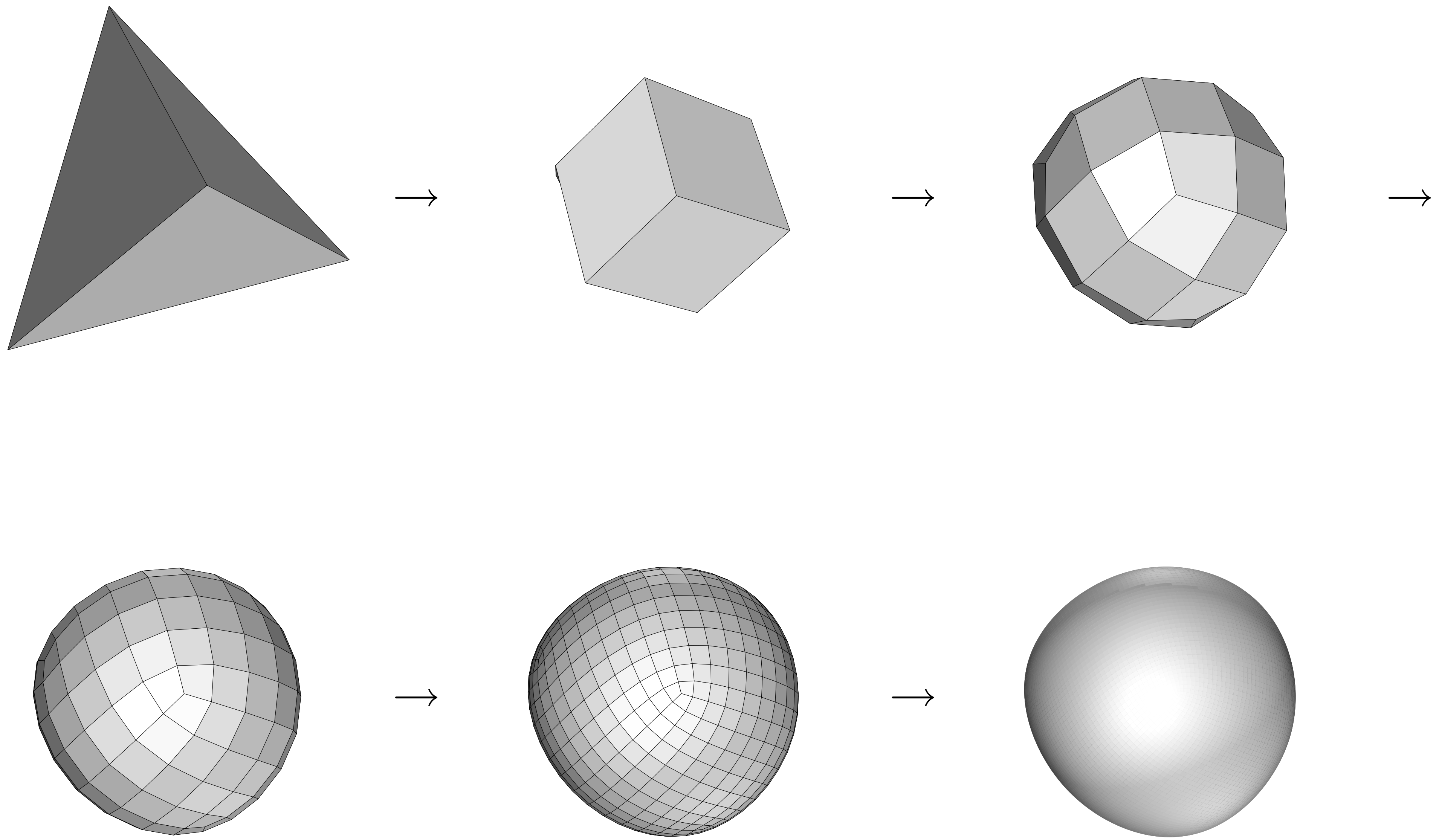


Edge Point (Boundary)

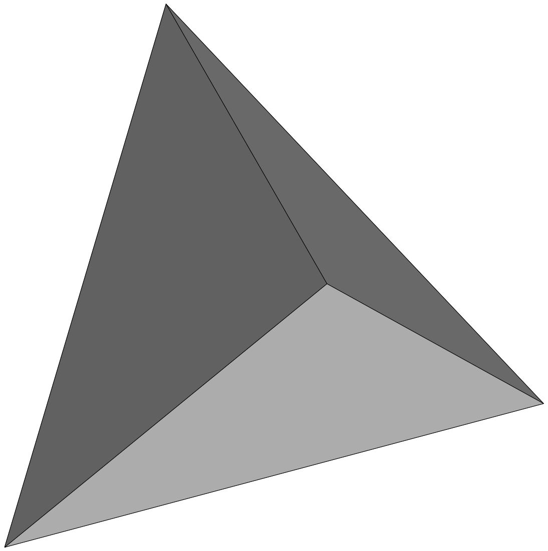


Old Vertex (Boundary)

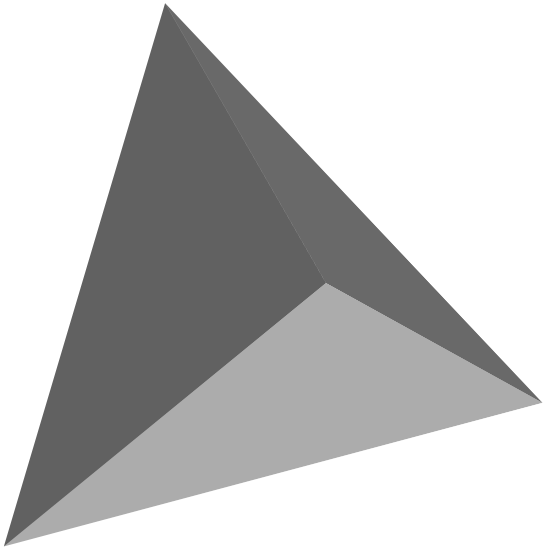
Catmull-Clark Subdivision: Example



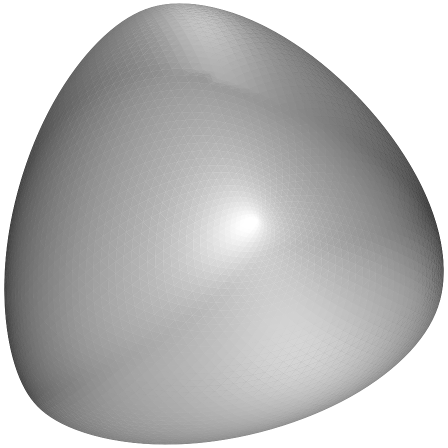
Comparison of Limit Surfaces for Various Schemes



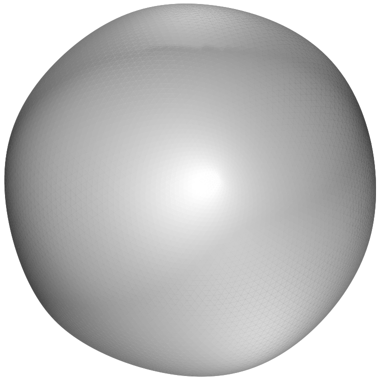
Control Polyhedron



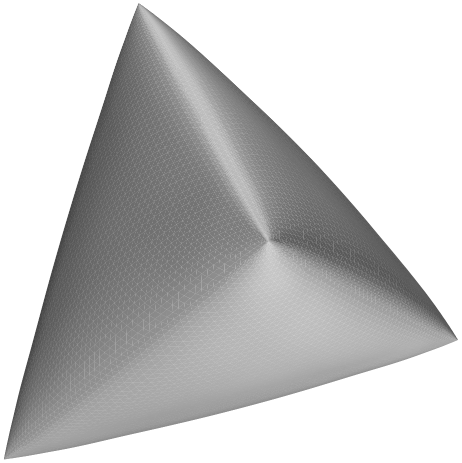
Linear



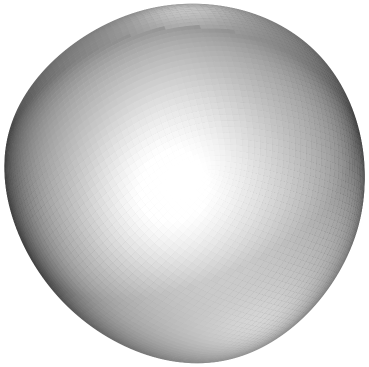
Loop



Kobbelt $\sqrt{3}$



Butterfly



Catmull-Clark

Section 6.1.2

More on Subdivision

Subdivision and Affine Invariance

- recall, for any affine transformation T and any finite set $\{p_i\}$ of points,
$$T\left(\sum_i a_i p_i\right) = \sum_i a_i T(p_i) \quad \text{where} \quad \sum_i a_i = 1$$
(i.e., affine transformations ***commute*** with affine combinations)
- let M denote polygon mesh, S denote subdivision operator, and T denote affine transformation
- subdivision having property of ***affine invariance*** means
$$T\left(S(M)\right) = S\left(T(M)\right)$$
(i.e., applying affine transformation T followed by subdivision yields same result as applying subdivision followed by affine transformation T)
- affine invariance ***extremely desirable*** (in practical sense, means subdivision result coordinate-system independent)
- in order for subdivision to be affine invariant, geometric refinement rules must employ masks that correspond to ***affine combinations*** (i.e., mask coefficients sum to one)
- not coincidence that all subdivision methods discussed have geometric refinement masks whose coefficients sum to one

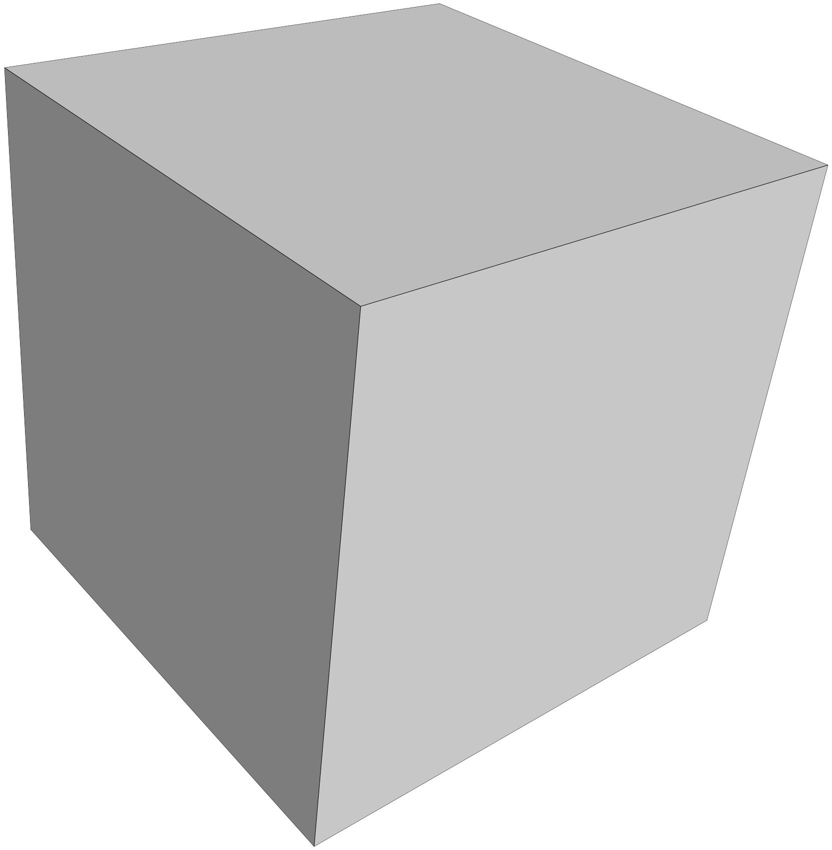
Subdivision and Boundaries

- if geometric refinement masks for boundary vertices ***depend only on boundary vertices***, border in refined mesh is function of only boundary vertices in control mesh
- suppose mesh M cut into two along some subset E of its edges to form new meshes M_1 and M_2 and then subdivision applied independently to each of M_1 and M_2 to produce refined meshes M_1' and M_2', respectively
- if subdivision scheme such that geometric refinement of boundary vertices depends only on boundary vertices, then M_1' and M_2' must ***fit together perfectly*** (except possibly near start/end of cut) (i.e., have positional continuity) along boundary corresponding to cut (along edge set E)
- above property of positional continuity is ***highly desirable*** in many situations
- not coincidence that all subdivision schemes considered are such that geometric refinement of boundary vertices depends only on boundary vertices
- above property of positional continuity along boundary can be exploited to generate surfaces with ***creases***

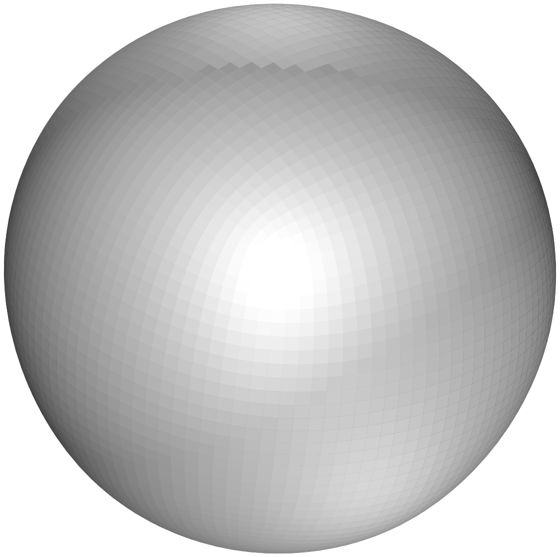
Surfaces With Creases

- often, want to represent surface that is mostly smooth, but has sharp creases along some specific edges
- treat crease edges as boundary edges
- since masks for boundary cases normally do not depend on non-boundary vertices positional (i.e., G_0) continuity maintained
- ensures positional continuity but will generally tend not to be smooth
- in this way, creases in surface can be generated

Crease Example



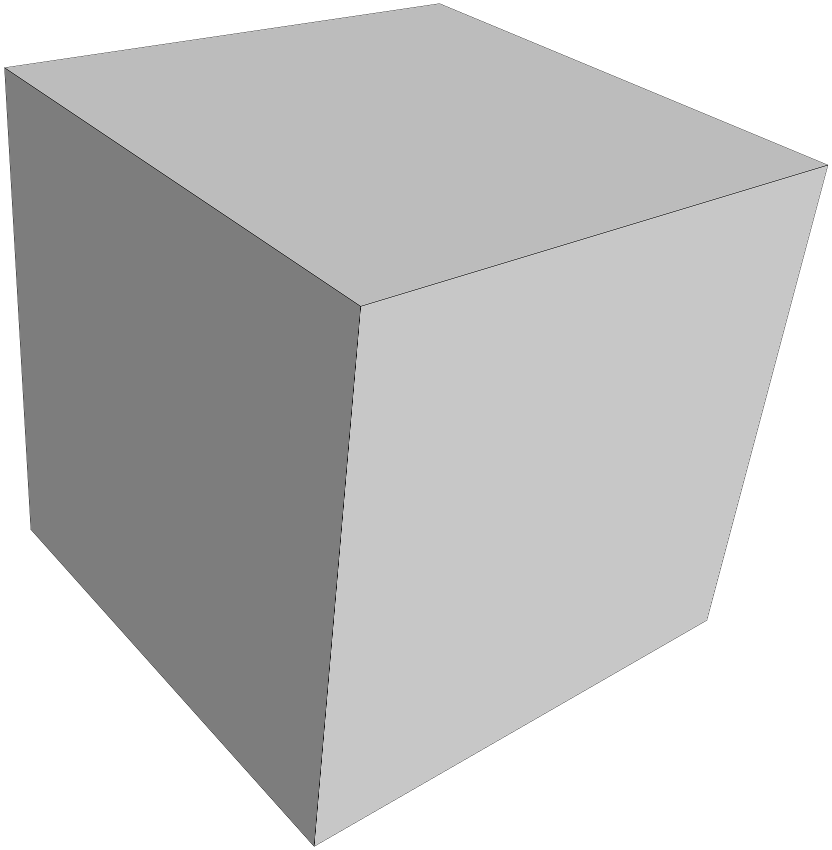
Control Mesh



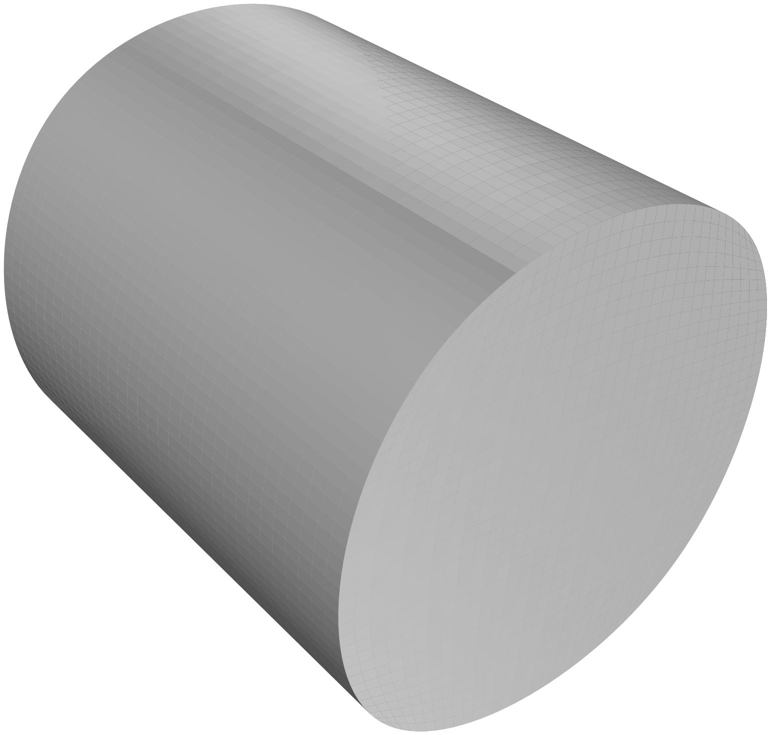
Limit Surface

- subdivision with no sharp edges (i.e., classic subdivision)

Crease Example (Continued 1)



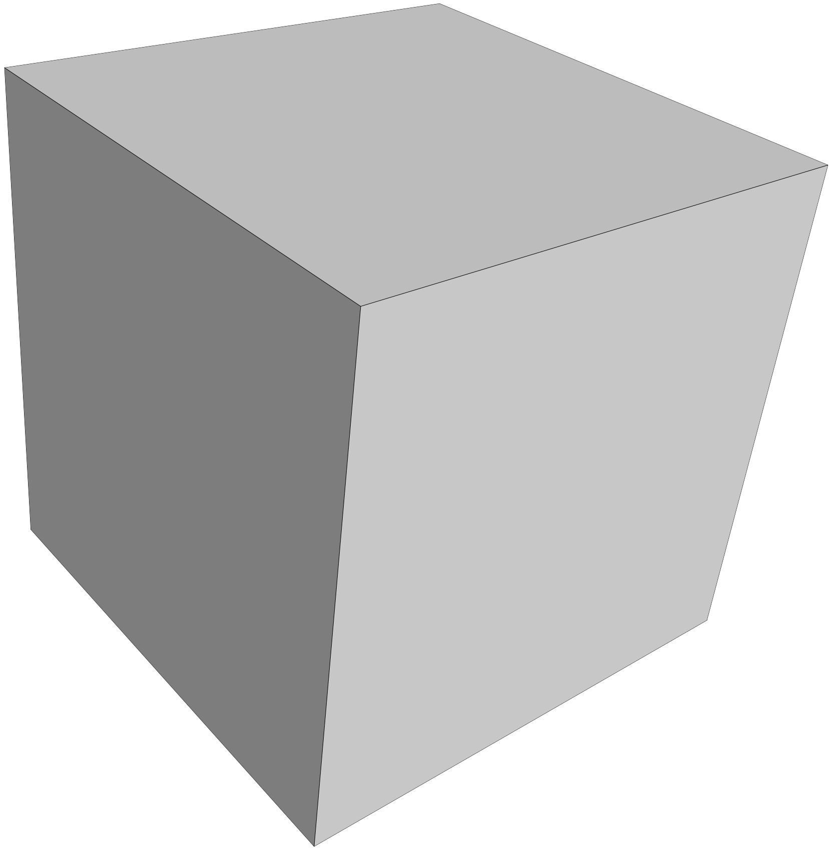
Control Mesh



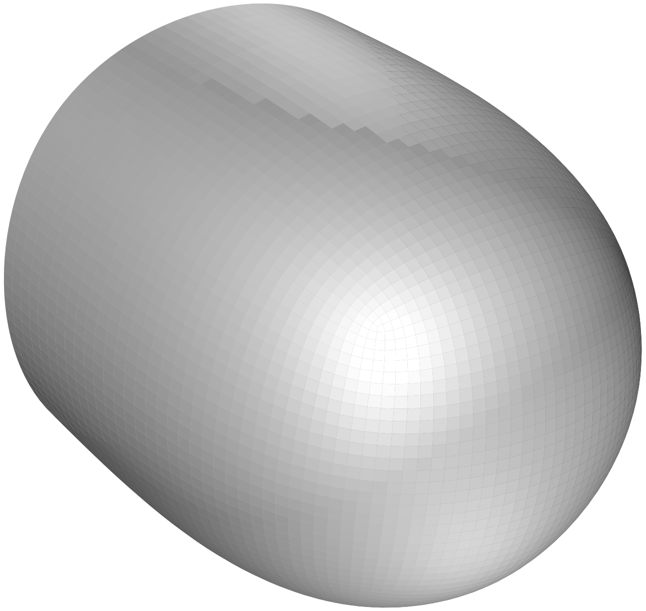
Limit Surface

- subdivision with all edges from two opposing faces marked as sharp

Crease Example (Continued 2)



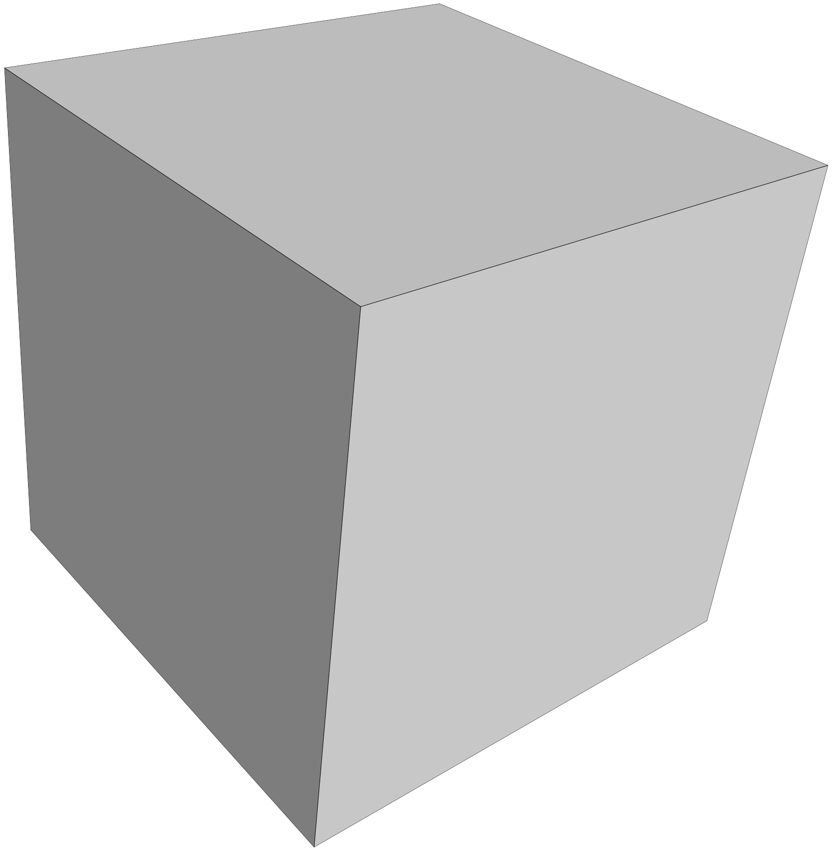
Control Mesh



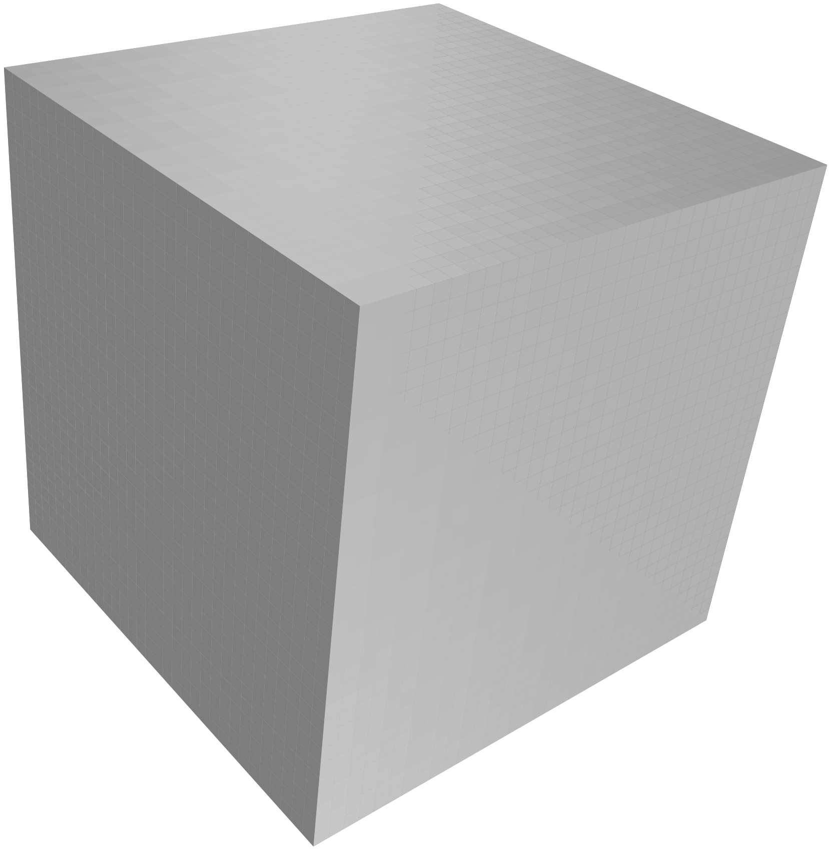
Limit Surface

- subdivision with all four edges of single face marked as sharp

Crease Example (Continued 3)



Control Mesh



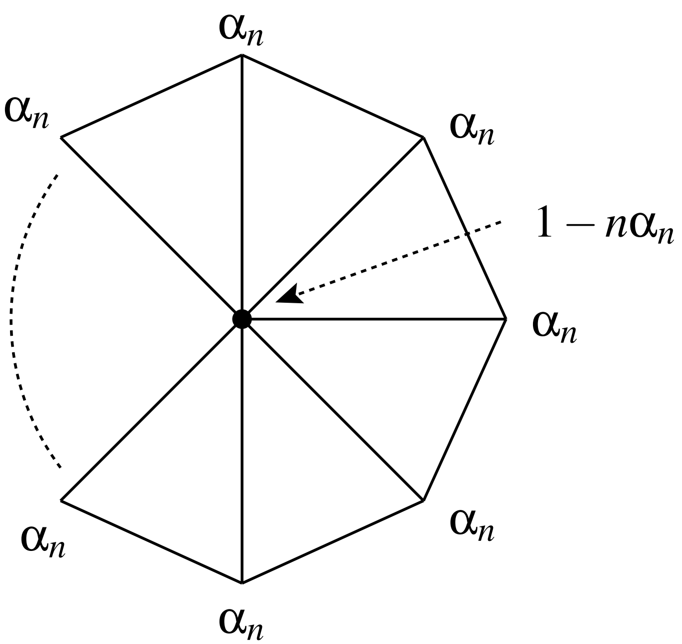
Limit Surface

- subdivision with all edges in control mesh marked as sharp

Position and Tangent Masks

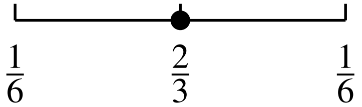
- In the case of an approximating scheme, each iteration of subdivision results in a repositioning of old vertices.
- For a particular vertex in the mesh, we might want to determine where the vertex will be positioned in the limit (i.e., after an infinite number of iterations of subdivision are applied).
- As it turns out, the limit position of a vertex can be easily found by taking an appropriate linear combination of nearby mesh vertices. This linear combination can be represented pictorially as a ***position mask***.
- Similarly, we might want to determine a tangent vector to the limit surface at the limit position of a vertex.
- A tangent vector can also be computed using an appropriate linear combination of nearby mesh vertices. This linear combination can be represented pictorially as a ***tangent mask***.
- Using two tangent masks, we can determine two tangent vectors, which can then be used to calculate a surface normal (via a vector cross product).

Loop Subdivision: Position Masks



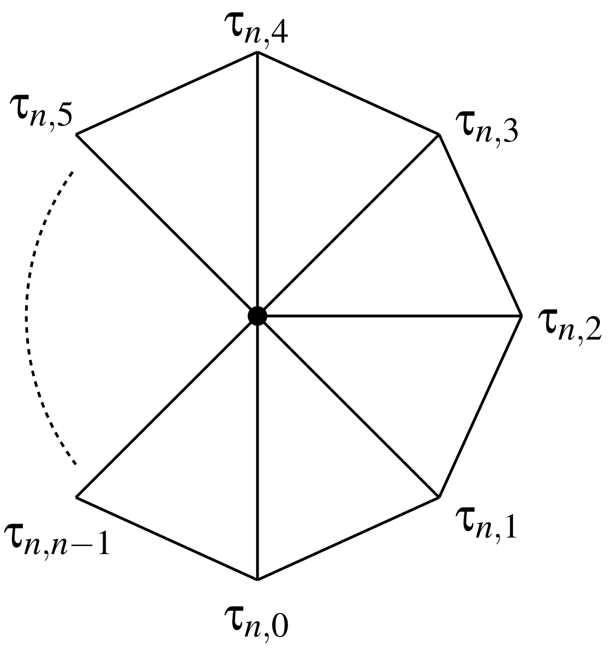
$$\alpha_n = \left(\frac{3}{8\beta_n} + n\right)^{-1}$$

Interior

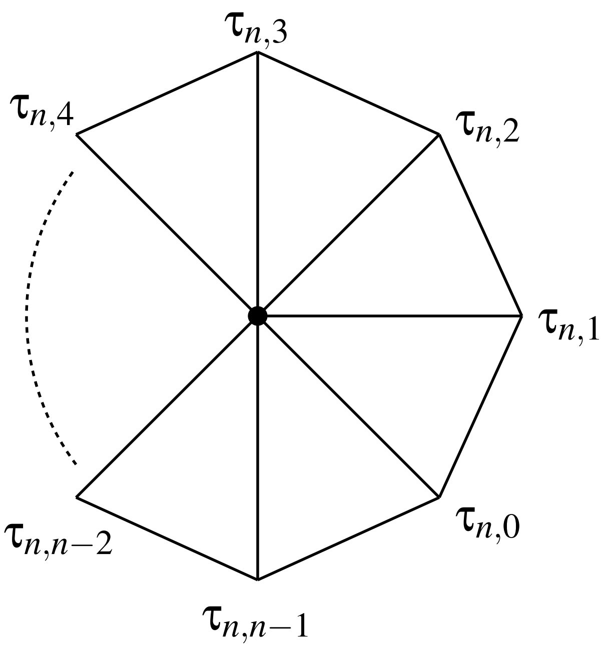


Boundary

Loop Subdivision: Tangent Masks



$$\tau_{n,k} = \cos \frac{2\pi k}{n}$$



$$\tau_{n,k} = \cos \frac{2\pi k}{n}$$

Section 6.2

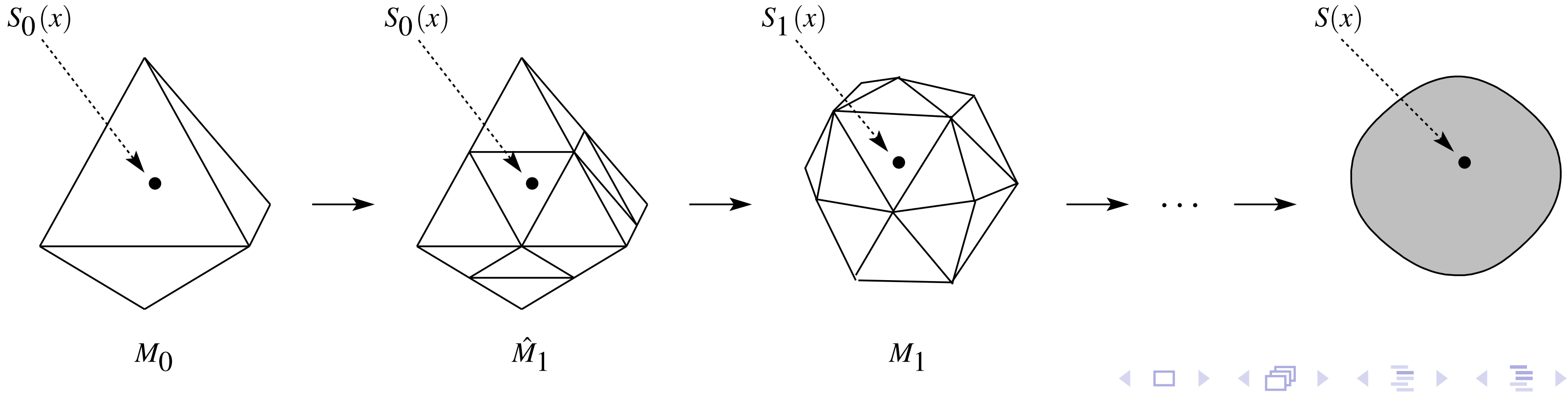
Subdivision Wavelets

Primal Subdivision as Splitting and Averaging

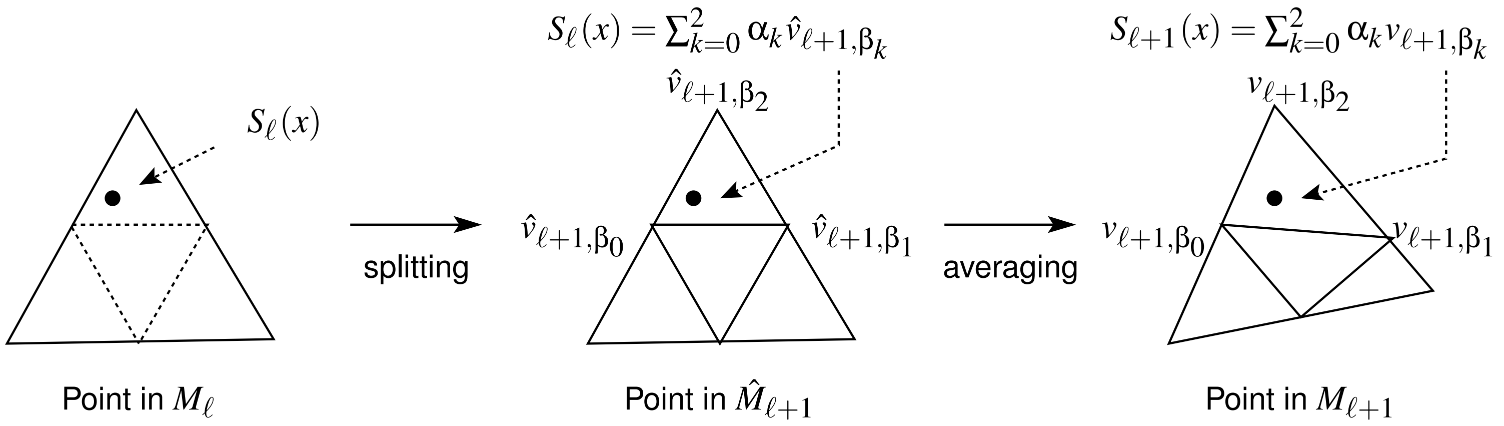
- Subdivision iteratively refines a control polyhedron M_0 to produce increasingly faceted polyhedra $M_1, M_2, \ldots$ that converge to the limit surface M_∞.
- In each subdivision step, the vertices of $M_{\ell+1}$ are computed as affine combinations of the vertices of M_ℓ.
- The refinement process that transforms M_ℓ to $M_{\ell+1}$ can be viewed as consisting of ***two steps***: 1) splitting and 2) averaging.
- In the ***splitting step***, each face of M_ℓ is split into k new faces, yielding the intermediate mesh $\hat{M}_{\ell+1}$. In the case of primal triangle quadrisection, a face is split into $k = 3$ new faces by introducing a new vertex at the midpoint of each edge.
- In the ***averaging step***, the vertex positions of $M_{\ell+1}$ are computed as affine combinations of the vertices of $\hat{M}_{\ell+1}$.
- Using this splitting and averaging view of subdivision, we can show that M_∞ can be expressed parametrically using M_0 as the domain of the parameterization.

Subdivision-Surface Parameterization: Basic Idea

- We establish a correspondence between the points on M_ℓ and $M_{\ell+1}$.
- Then, we use this correspondence to track the movement of an arbitrary point originating on M_0 from one subdivision level to the next, as subdivision is applied repeatedly.
- As the number of iterations approaches infinity, the point converges to a point on the limit surface.
- The above process establishes a correspondence between points on M_0 and points on the limit surface M_∞.
- The parametric representation of M_∞ immediately follows from this mapping from a point originally on M_0 to a point on M_∞.



Subdivision-Surface Parameterization: Point Correspondence


$S_\ell(x)$
Point in M_ℓ
splitting
$S_\ell(x) = \sum_{k=0}^{2} \alpha_k \hat{v}_{\ell+1,\beta_k}$
$\hat{v}_{\ell+1,\beta_2}$
$\hat{v}_{\ell+1,\beta_0}$
$\hat{v}_{\ell+1,\beta_1}$
Point in $\hat{M}_{\ell+1}$
averaging
$S_{\ell+1}(x) = \sum_{k=0}^{2} \alpha_k v_{\ell+1,\beta_k}$
$v_{\ell+1,\beta_2}$
$v_{\ell+1,\beta_0}$
$v_{\ell+1,\beta_1}$
Point in $M_{\ell+1}$

- Suppose that $S_\ell(x)$ lies in triangle $(\hat{v}_{\ell+1,\beta_0}, \hat{v}_{\ell+1,\beta_1}, \hat{v}_{\ell+1,\beta_2})$ of $\hat{M}_{\ell+1}$ with barycentric coordinates $(\alpha_0, \alpha_1, \alpha_2)$. That is,

$$S_\ell(x) = \sum_{k=0}^{2} \alpha_k \hat{v}_{\ell+1,\beta_k}.$$

- Then, a corresponding point $S_{\ell+1}(x)$ in $M_{\ell+1}$ can be chosen as

$$S_{\ell+1}(x) = \sum_{k=0}^{2} \alpha_k v_{\ell+1,\beta_k}.$$

- Note that $\{\hat{v}_{\ell+1,\beta_k}\}_{k=0}^{2}$ and $\{v_{\ell+1,\beta_k}\}_{k=0}^{2}$ are not generally the same except for linear subdivision.
- The above establishes a correspondence between points on M_ℓ and $M_{\ell+1}$.

Subdivision-Surface Parameterization

- We select

$$S_0(x) = x \text{ for } x \in M_0.$$

- Then, using the relationship between $S_\ell(x)$ and $S_{\ell+1}(x)$ from the previous slide, we can express M_∞ parametrically as

$$S(x) = \lim_{\ell \to \infty} S_\ell(x).$$

- Observe that the domain of the above parameterization is M_0.
- As we shall see, the above subdivision-surface parameterization leads to a wavelet representation for meshes.

Approximation Spaces and Scaling Functions

- **Theorem.** For any subdivision scheme, and for any $\ell \geq 0$, given the vertices $\{v_{\ell,k}\}$ of the mesh M_ℓ, there exist scalar-valued functions $\{\phi_{\ell,k}\}$ defined on M_0 such that

$$S(x) = \textstyle\sum_k v_{\ell,k}\phi_{\ell,k}(x).$$

- The $\{\phi_{\ell,k}\}$ are called **scaling functions**.
- **Theorem.** The scaling functions $\phi_{\ell,k}$ are ***refinable***, in the sense that each $\phi_{\ell,k}$ can be expressed in terms of $\{\phi_{\ell+1,k}\}$ as

$$\phi_{\ell,k} = \textstyle\sum_k a_k\phi_{\ell+1,k},$$

  where $\{a_k\}$ is a scalar sequence.
- From the scaling functions, we can define a sequence $\{V_\ell\}$ of spaces, called **approximation spaces**, as

$$V_\ell = \operatorname{span}\{\phi_{\ell,k}\}.$$

- Using the refinability of the scaling functions, we can show that the $\{V_\ell\}$ are nested as

$$V_0 \subset V_1 \subset V_2 \subset \ldots.$$

Wavelets Spaces and Wavelet Functions

- Since $V_\ell \subset V_{\ell+1}$, there must exist some subspace W_ℓ of $V_{\ell+1}$ such that $V_{\ell+1} = V_\ell \oplus W_\ell$.
- Thus, we can associate with $\{V_\ell\}$ another sequence of spaces $\{W_\ell\}$.
- The $\{W_\ell\}$ are called **wavelet spaces**.
- For each wavelet space W_ℓ, we can find a basis $\{\psi_{\ell,k}\}$.
- The $\{\psi_{\ell,k}\}$ are called **wavelet functions**.

Wavelet Analysis and Synthesis Operators

- A surface $s \in V_\ell$ can be represented in two different ways.
- Since $s \in V_\ell$, s has an expansion in terms of the basis of V_ℓ given by
$$s(x) = \sum_k v_{\ell,k} \phi_{\ell,k}(x). \tag{1}$$
- Furthermore, as $V_\ell = V_{\ell-1} \oplus W_{\ell-1}$, we can also expand s in terms of the bases for $V_{\ell-1}$ and $W_{\ell-1}$ to obtain
$$s(x) = \sum_k v_{\ell-1,k} \phi_{\ell-1,k}(x) + \sum_k w_{\ell-1,k} \psi_{\ell-1,k}(x). \tag{2}$$
- In (2), the first summation corresponds to a ***coarse approximation*** of s that lies in $V_{\ell-1}$, while the second summation is associated with ***fine detail*** (missing from the coarse approximation) that lies in $W_{\ell-1}$.
- The $\{w_{\ell,k}\}$ are called **wavelet coefficients**.
- Note that the $\{v_{\ell,k}\}$ and $\{w_{\ell,k}\}$ are vectors.
- The transformation from (1) to (2) is a ***wavelet analysis operator***.
- The transformation from (2) to (1) is a ***wavelet synthesis operator***.
- Subdivision is essentially equivalent to a wavelet synthesis operator with all wavelet coefficients set to the zero vector.

Mallat Algorithm

- ***Wavelet analysis operator:*** Given $\{v_{\ell,k}\}$, we can compute the corresponding $\{v_{\ell-1,k}\}$ and $\{w_{\ell-1,k}\}$ by filtering operations.
- ***Wavelet synthesis operator:*** Given $\{v_{\ell-1,k}\}$ and $\{w_{\ell-1,k}\}$, we can compute the corresponding $\{v_{\ell,k}\}$ by a filtering operation.

Section 6.3

Applications of Subdivision Surfaces and Wavelets

Applications of Subdivision Surfaces

- Since subdivision is essentially an interpolation process for surfaces, it can be used in any application that needs to model surfaces.
- Some applications of subdivision surfaces include:
  - multimedia
  - animation
  - gaming
  - biomedical computing
  - computer-aided design and manufacturing
  - geometric modelling
  - finite element analysis
  - computational fluid dynamics
  - scientific visualization

Applications of Subdivision Wavelets

- Subdivision wavelets provide a multiresolution representation for surfaces of arbitrary topological type.
- Therefore, any application that must deal with surfaces can potentially benefit from subdivision wavelets.
- Some applications of subdivision wavelets include:
  - polygon mesh compression
  - continuous level-of-detail control
  - compression of functions defined on surfaces
  - multiresolution editing of surfaces
  - surface optimization
  - numerical solution of integral and differential equations (involving functions defined on surface of arbitrary topological type)

Part 7

Applications in Signal Processing

Section 7.1

Signal Coding

Signal Coding

- **signal coding**: seek alternative representations of signals with some specific purpose in mind
- **signal compression**: goal to use representation of signal with less redundancy so that fewer bits required to specify signal
- coding can be either lossless or lossy
- **lossless**: no information loss (i.e., reconstructed signal identical to original signal)
- **lossy**: information loss (i.e., reconstructed signal only approximation to original)
- less space for storage (e.g., on disk or in memory) and less bandwidth/time for transmission
- two general approaches to signal compression: 1) time/spatial-domain coding and 2) transform coding
- **time/spatial-domain coding**: samples of signal coded directly
- transform coding case to be discussed shortly

Measuring Coding Efficiency

- **rate**: measure of number of bits needed to represent signal
- **distortion**: measure of approximation error in reconstructed (i.e., decoded) signal
- for any given rate, want to minimize distortion
- generally, distortion tends to increase as rate decreases
- rate can be expressed in number of ways
- **compression ratio**: ratio of original signal size in bits to compressed signal size in bits
- **normalized bit rate**: reciprocal of compression ratio
- **bit rate**: compressed signal size (in bits/sample)
- many distortion measures possible
- often distortion measured by mean-squared error (MSE)
- in case of lossless coding, distortion always zero

Mean-Squared Error and Peak-Signal-to-Noise Ratio

- For a signal x and its reconstruction $\tilde{x}$ sampled at the points in Λ, the **mean-squared error (MSE)** is defined as

$$\text{MSE} = |\Lambda|^{-1} \sum_{k \in \Lambda} \left(\tilde{x}(k) - x(k)\right)^2 .$$

- The MSE is typically expressed in terms of the **peak-signal-to-noise ratio (PSNR)** as defined by

$$\text{PSNR} = 20 \log_{10} \left(\frac{M}{\sqrt{\text{MSE}}} \right),$$

  where M is the theoretical maximum absolute error. (For integer data, $M = 2^P - 1$, where P is the number of bits/sample).

- An ***increase*** in the PSNR (in dB) by Δp corresponds to a ***decrease*** in the MSE by the (multiplicative) factor $10^{\Delta p/10}$. (For example, a 1 dB increase in the PSNR corresponds to a reduction in the MSE by a factor of $10^{1/10} \approx 1.2589$.)

- Note that, as the MSE increases, the PSNR decreases.

Transform/Subband Coding

- with transform coding, transform applied to samples of signal to be coded
- then, resulting transform coefficients coded
- transform employed in attempt to obtain data that is easier to code efficiently
- commonly used for lossy coding
- reversible integer-to-integer transforms growing in popularity for lossless coding
- for lossy coding, transform/subband coders tend to have better coding efficiency for fixed complexity

Transform/Subband Coding

- quantization discards transform-coefficient information deemed to be insignificant
- in case of lossless coding, quantization is not performed
- entropy coding is lossless
- entropy coding exploits statistical model of data in order to code symbols with higher probability using fewer bits
- only potential for information loss due to quantization (assuming that transform in encoder and decoder are exact inverses, even in finite-precision arithmetic)

Uniform Scalar Quantization

- midtread uniform scalar quantization rounds real number to integer multiple of some unit of precision called step size
- quantization essentially represents real number x by approximation $Q(x)$, where

$$Q(x) = \Delta(\operatorname{sgn} x)\left\lfloor \frac{|x|}{\Delta} + \frac{1}{2} \right\rfloor$$

- quantization step size Δ controls granularity (i.e., coarseness) of rounding (e.g., if $\Delta = 1$, quantization rounds to nearest integer)
- classification rule: maps real number x to integer quantization index k as given by

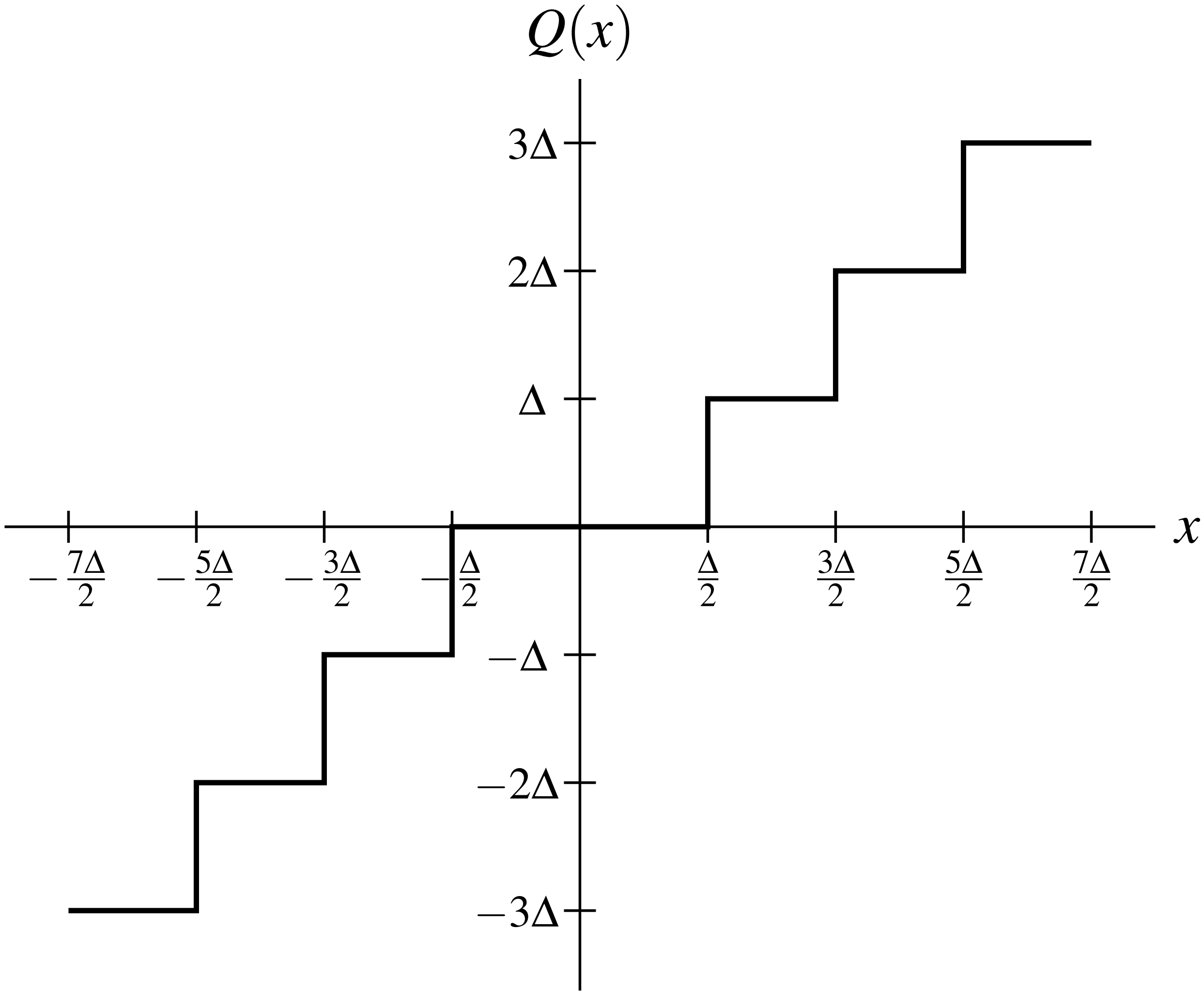
$$k = \frac{Q(x)}{\Delta} = (\operatorname{sgn} x)\left\lfloor \frac{|x|}{\Delta} + \frac{1}{2} \right\rfloor$$

- reconstruction rule: maps integer quantization index k to real reconstruction value y as given by

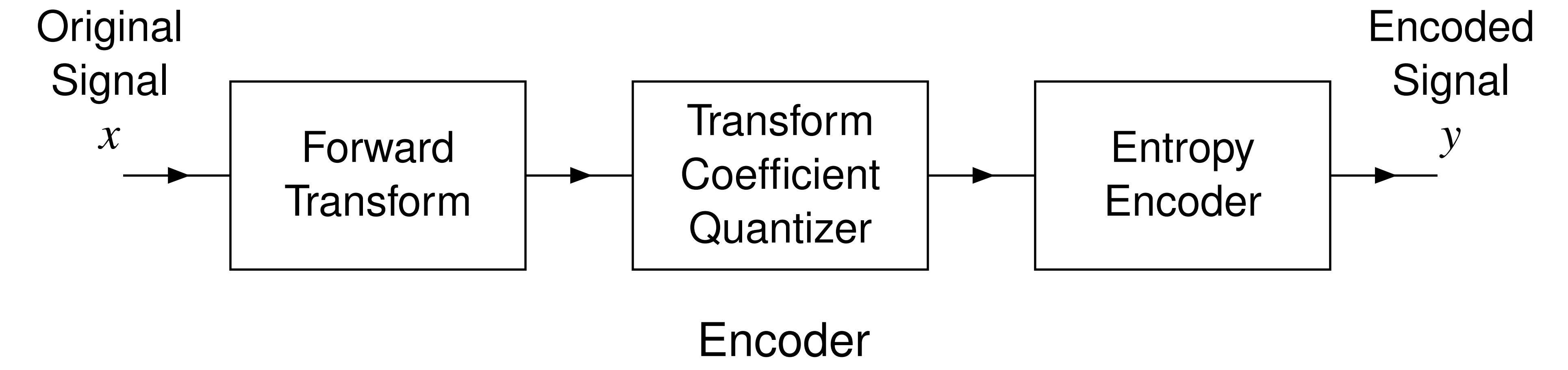
$$y = Q(x) = k\Delta$$

- for example: if $x = 5$ and $\Delta = 4$, then $k = \left\lfloor \frac{5}{4} + 1/2 \right\rfloor = \lfloor 7/4 \rfloor = 1$ and $y = 1(4) = 4$

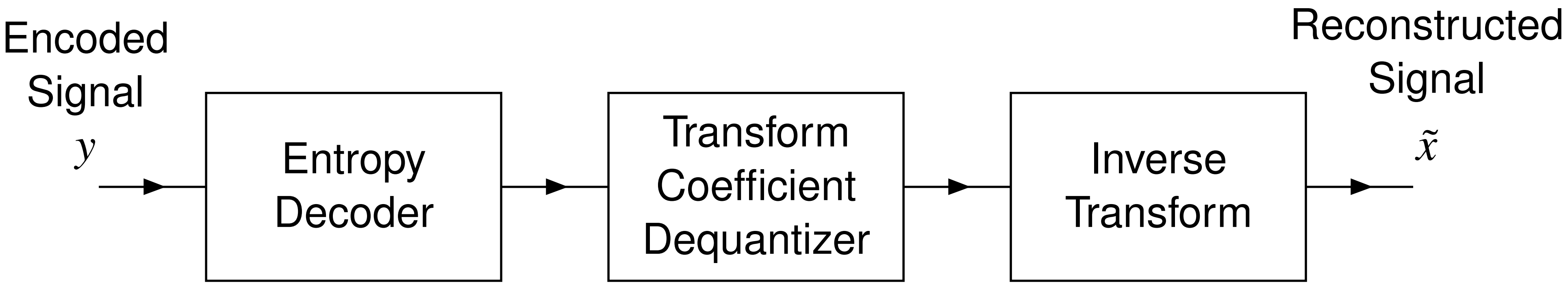
Uniform Scalar Quantization Transfer Characteristic



Structure of Transform/Subband Coder



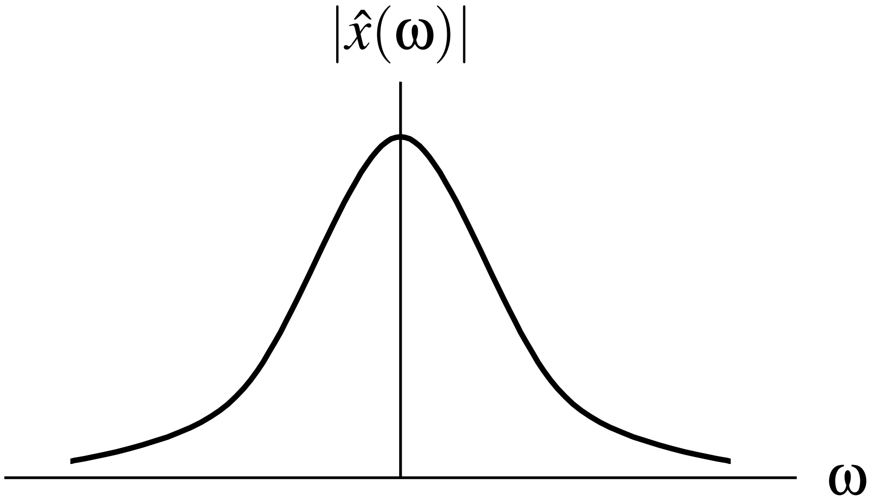
Encoder



Decoder

Spectrum of Typical Signal

- typically, most signals tend to be lowpass in nature
- most information at low frequencies
- amount of information decreases rapidly as frequency increases
- information not uniformly distributed across all frequencies
- example:



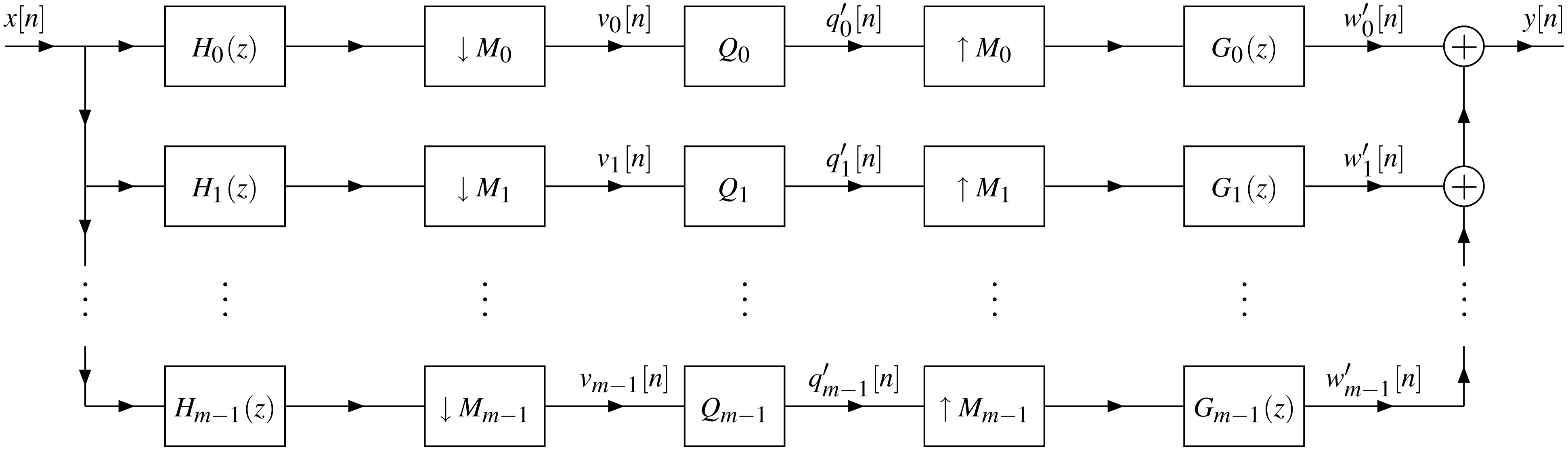
Subband Coding

- forward and inverse transforms are analysis and synthesis sides of PR maximally-decimated filter bank, respectively
- PR system employed so that transform does not itself introduce distortion (except possibly due to finite-precision effects)
- subband transform decomposes signal into frequency bands
- exploit nonuniform distribution of energy in spectrum of signal
- not all subband signals have same energy content
- for example, for transform derived from one-dimensional m-channel UMD filter bank with ideal frequency-selective filters, if original signal bandlimited to baseband frequencies in range $[-\frac{\pi}{m}, \frac{\pi}{m}]$, only 0th channel would have any energy content; thus, only $\frac{1}{m}$ of samples need to be coded
- in practice, however, most subbands have some energy content, but not evenly distributed; typically lower frequency bands have most of energy

Subband Coding (Continued)

- subband coding used for many types of data such as speech, audio, image, video, and ECG
- for speech/audio/image/video, number of subbands, filter bandwidths, bit allocation chosen to exploit perceptual properties of human auditory/visual system

Model for Subband Coder



- m channels
- kth channel downsampled/upsampled by M_k
- analysis filters $\{H_k\}_{k=0}^{m-1}$
- synthesis filters $\{G_k\}_{k=0}^{m-1}$
- $\{Q_k\}_{k=0}^{m-1}$ quantizers

Subband Coding Gain

- interested in energy compacting ability of filter bank, which is often quantified by coding gain
- **coding gain** defined as ratio between reconstruction error variance obtained by quantizing signal directly to that obtained by quantizing corresponding subband coefficients using optimal bit-allocation strategy
- consider m-channel filter bank; let α_k denote reciprocal of downsampling factor in kth channel, $\{h_k\}_{k=0}^{m-1}$ denote analysis filter impulse responses, and $\{g_k\}_{k=0}^{m-1}$ denote synthesis filter impulse responses
- subband coding gain G_{SBC} given by

$$G_{\text{SBC}} = \prod_{k=0}^{m-1} \left(\frac{\alpha_k}{A_k B_k} \right)^{\alpha_k},$$

where

$$A_k = \sum_{l \in \mathbb{Z}^d} \sum_{p \in \mathbb{Z}^d} h_k[l] h_k[p] \rho_{xx}[p-l], \quad B_k = \alpha_k \sum_{l \in \mathbb{Z}^d} g_k^2[l],$$

and ρ_{xx} is autocorrelation of signal to be coded

Image Coding

- subband coding very commonly used for images
- two-dimensional filter banks employed
- most often, linear-phase FIR systems used
- frequently, octave-band filter bank (i.e., wavelet transform)
- wavelet transforms used in many coding systems, including JPEG-2000 image-compression standard (i.e., ISO/IEC 15444), FBI fingerprint-compression standard
- usually MSE used to measure distortion
- MSE does not always correlate well with distortion as perceived by human visual system

Subband Coding Gain Revisited

- The coding gain formula requires the assumption of a statistical model for the signal to be coded.
- In practice, for the case of images, we typically assume a first-order autoregressive (AR) model.
- Two variants of this model are commonly used: 1) separable and 2) isotropic.
- In these cases, in the formula for the subband coding gain, ρ_{xx} is given by

$$\rho_{xx}[n] = \begin{cases} \rho^{\|n\|_{l^1}} & \text{for separable model} \\ \rho^{\|n\|_{l^2}} & \text{for isotropic model,} \end{cases}$$

  where the correlation coefficient ρ satisfies $|\rho| \leq 1$.
- Typically, ρ is chosen to satisfy $\rho \in [0.90, 0.95]$.

Choice of Wavelet System

- separability preferred for reasons of computational efficiency
- orthogonality beneficial as it ensures numerical stability and facilitates selection of most important transform coefficients
- linear phase critical to avoid phase distortion (which can lead to badly distorted image edges)
- if orthogonality and linear phase not both possible, usually orthogonality is dropped
- high coding gain desirable

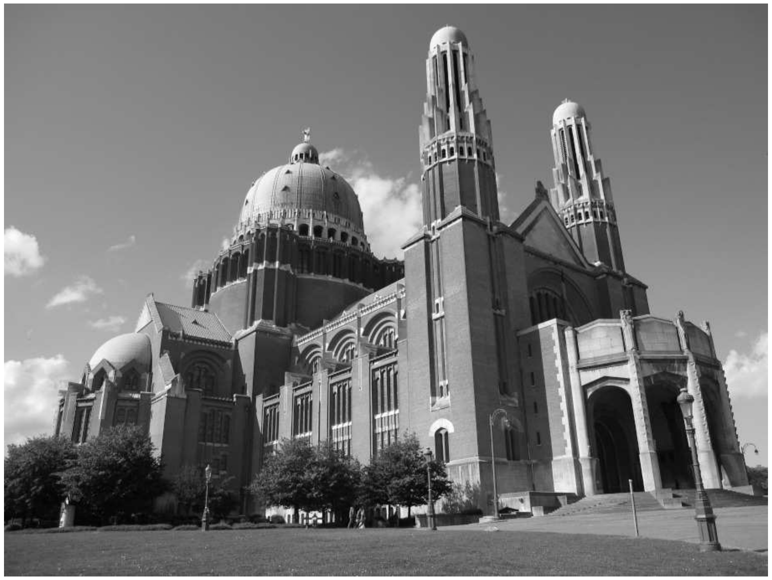
Characterizing Distortion

- In a transform/subband coder, we are representing the signal x to be coded as
$$x = \sum_{k \in I} a_k f_k,$$
where the $\{f_k\}_{k \in I}$ are the primal basis functions and the $\{a_k\}_{k \in I}$ are the transform coefficients.
- Suppose now that we quantize the transform coefficients, by replacing the $\{a_k\}_{k \in I}$ with their quantized versions $\{\tilde{a}_k\}_{k \in I}$, where $\tilde{a}_k = a_k + q_k$ and q_k corresponds to quantization error.
- In so doing, we obtain the quantized signal $\tilde{x}$, where
$$\tilde{x} = \sum_{k \in I} \tilde{a}_k f_k = \sum_{k \in I} (a_k + q_k) f_k.$$
- Thus, the error q introduced by quantization is given by
$$q = \sum_{k \in I} q_k f_k.$$
- In other words, the quantization error is a weighted sum of the primal basis functions $\{f_k\}_{k \in I}$.

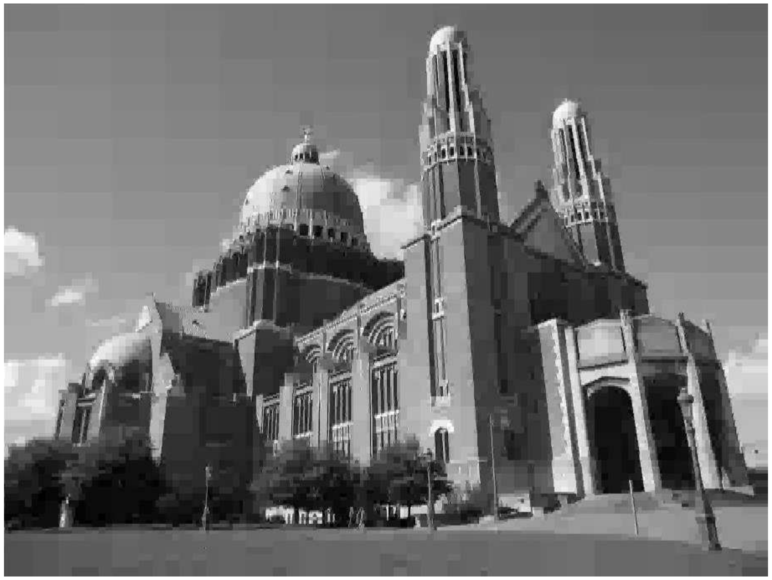
Characterizing Distortion (Continued)

- For a wavelet transform, the $\{f_k\}_{k \in I}$ are essentially sampled versions of approximations of the primal scaling and wavelet functions.
- Thus, the basis functions have a shape resembling the primal scaling and wavelet functions.
- Consequently, the quantization error q is approximately a weighted sum of translated and dilated versions of the primal scaling and wavelet functions.
- For this reason, the shape of the primal scaling and wavelet functions determine the nature of the artifacts introduced by compression.
- This point is illustrated in the coding example provided on the slides that follow.

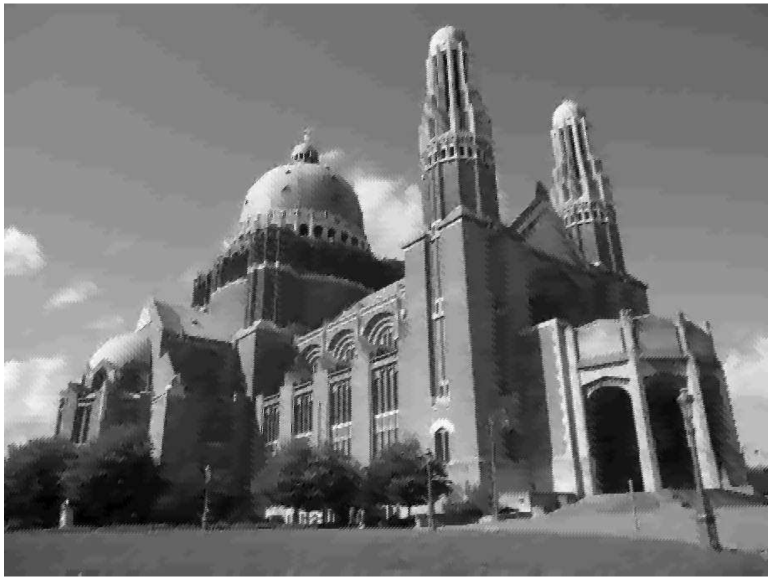
Image Coding Example (64:1 Compression)



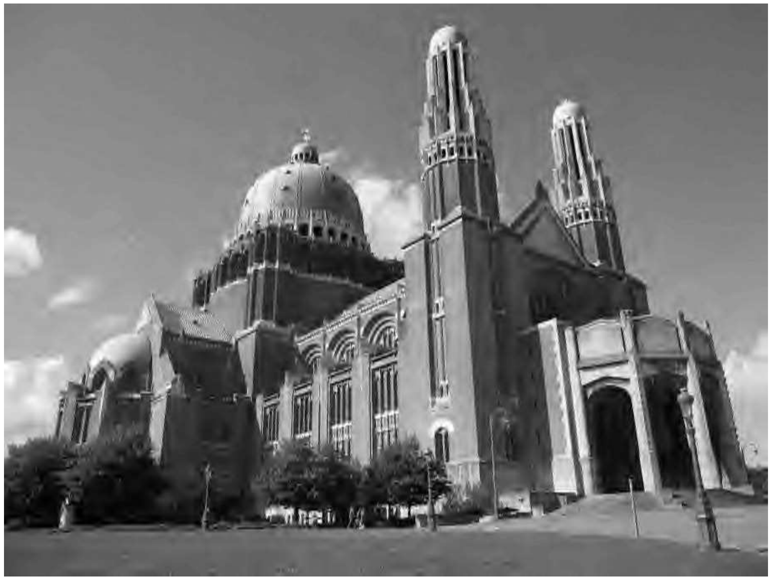
Original



Haar (26.55 dB)

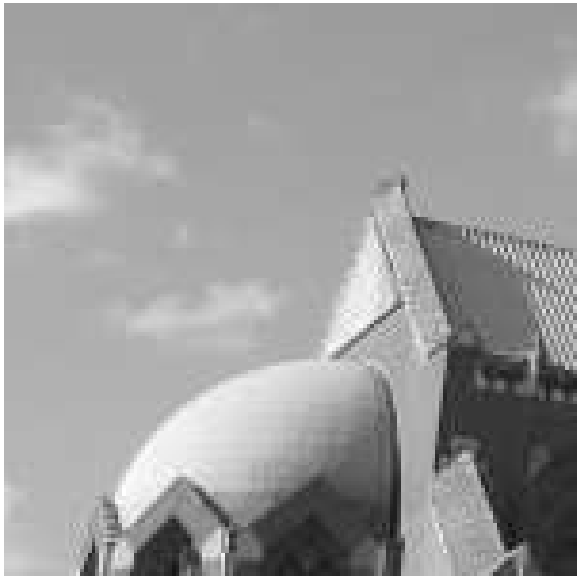


Twin Dragon (24.83 dB)

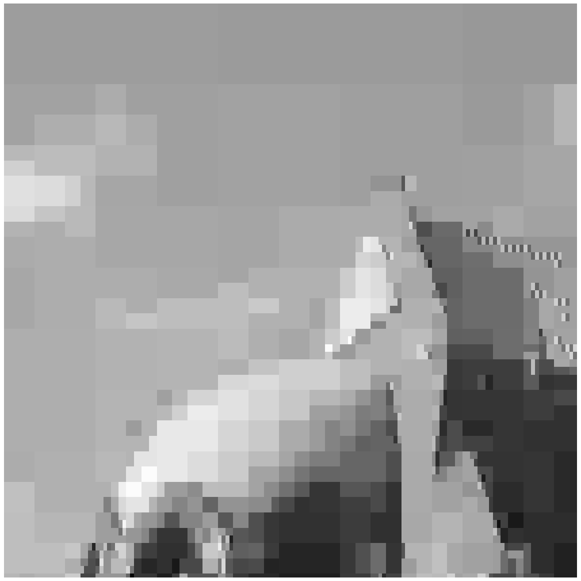


CDF 9/7 (27.68 dB)

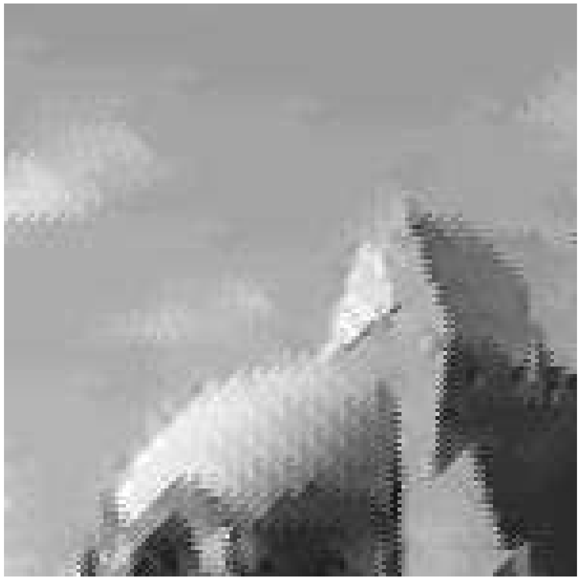
Image Coding Example (64:1 Compression) (Continued)



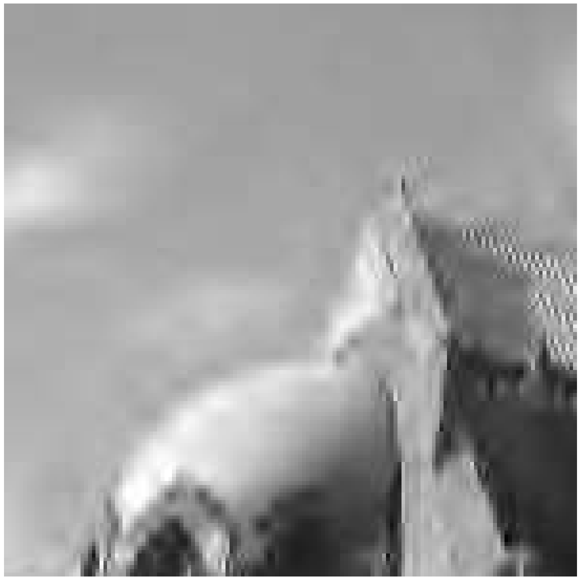
Original



Haar

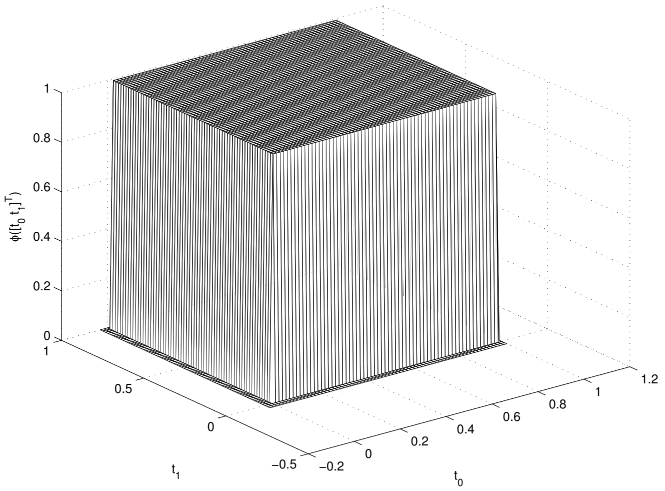


Twin Dragon

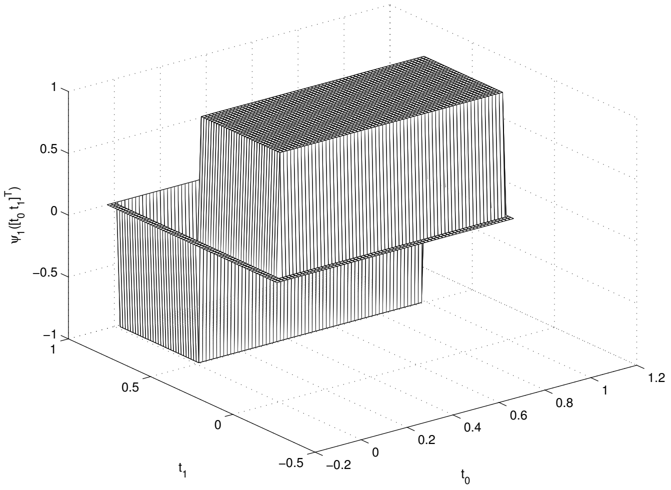


CDF 9/7

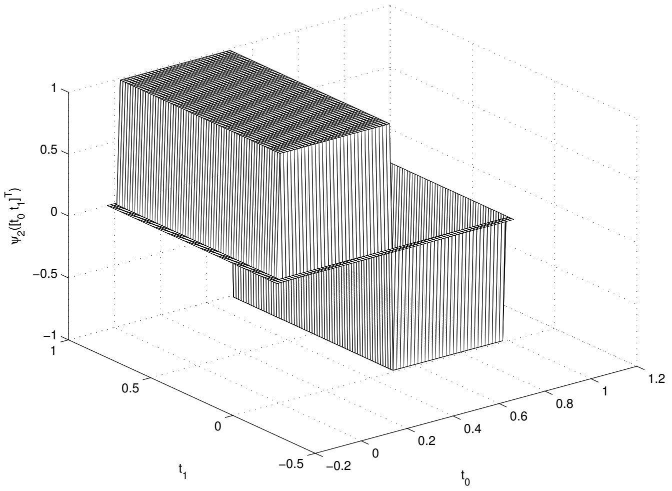
Haar Scaling and Wavelet Functions



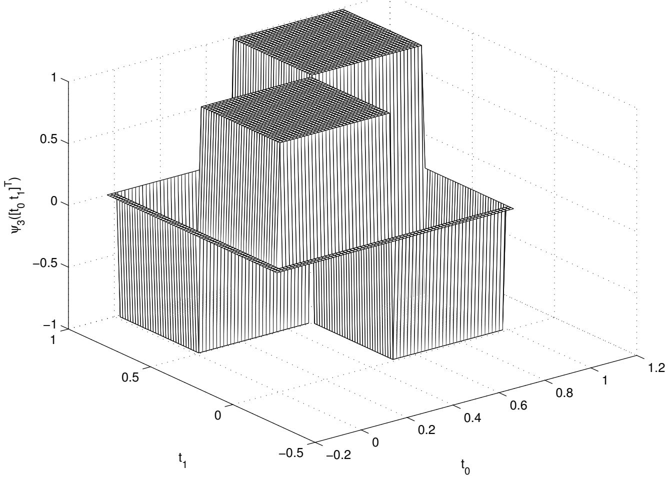
Scaling Function



Wavelet Function

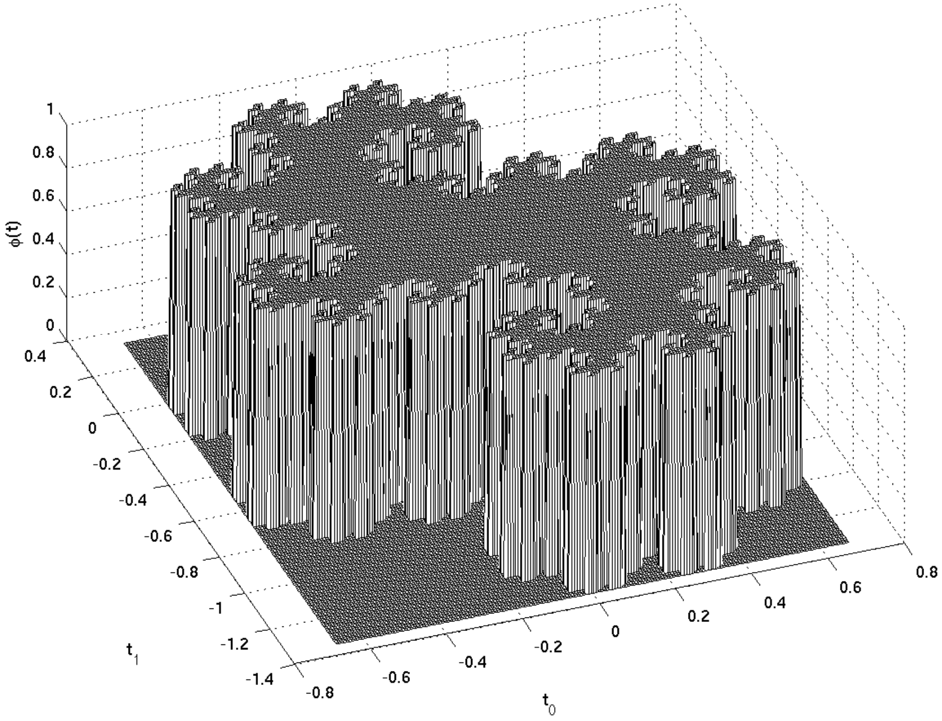


Wavelet Function

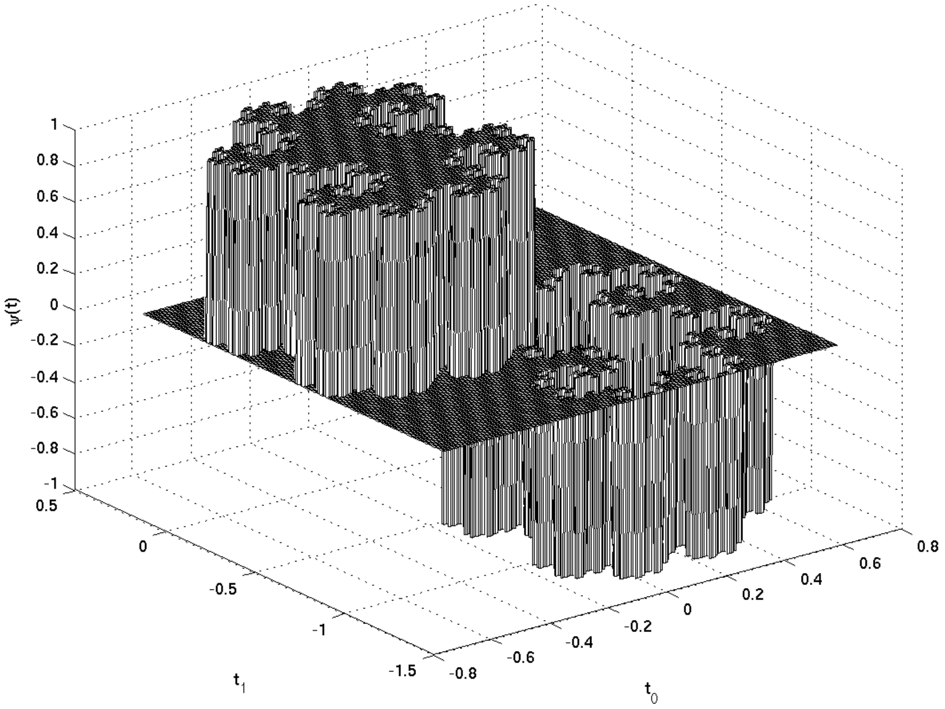


Wavelet Function

Twin-Dragon Scaling and Wavelet Functions

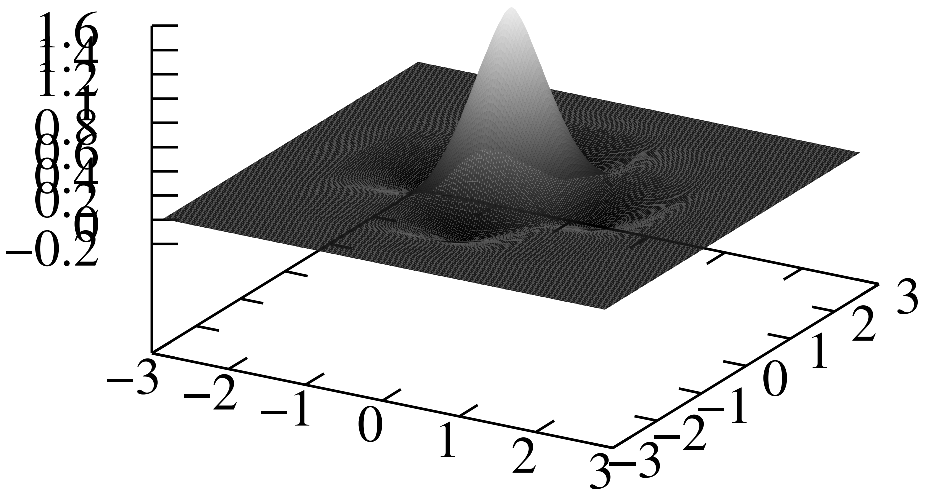


Scaling Function

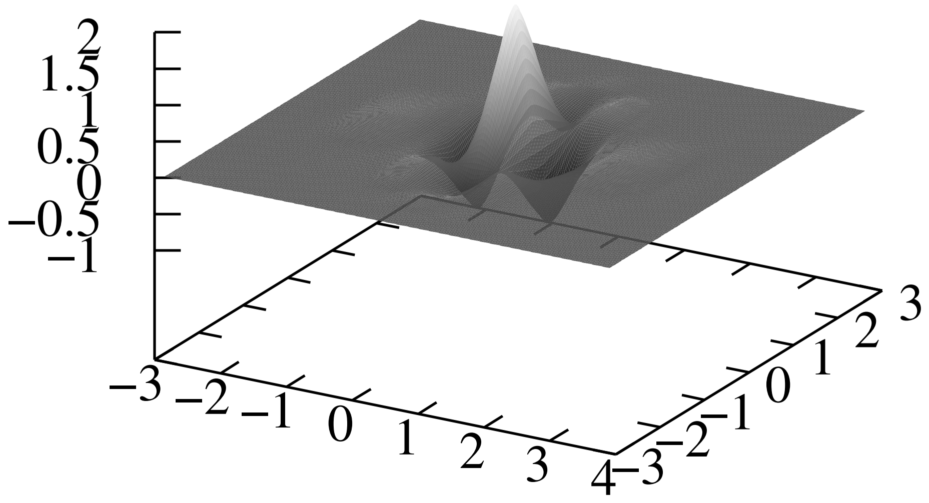


Wavelet Function

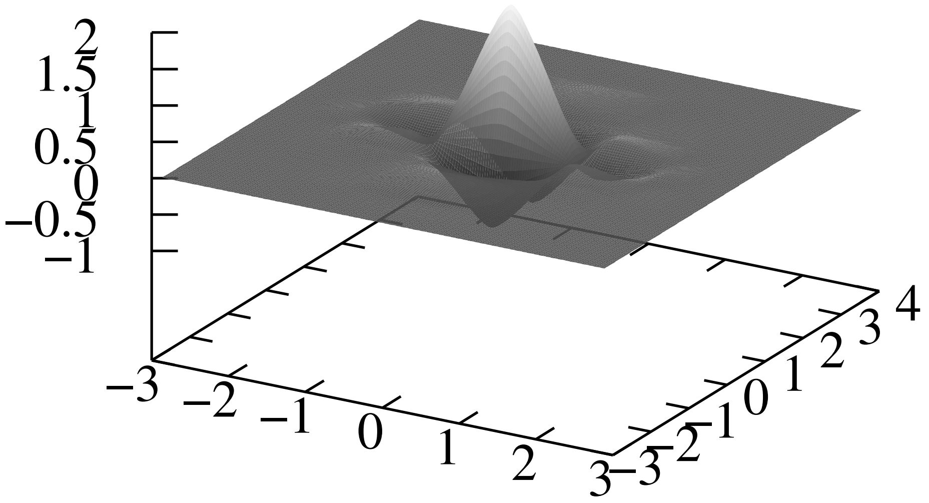
CDF 9/7 Primal Scaling and Wavelet Functions



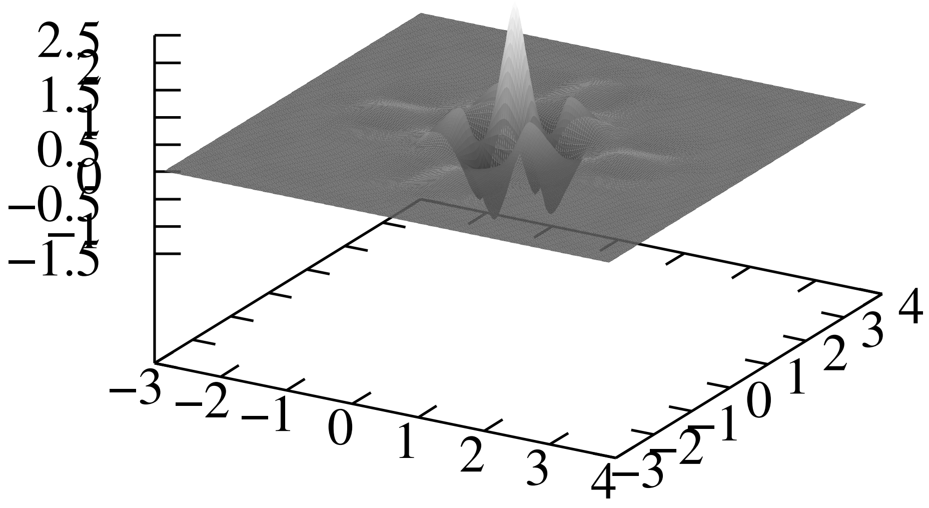
Scaling Function



Wavelet Function

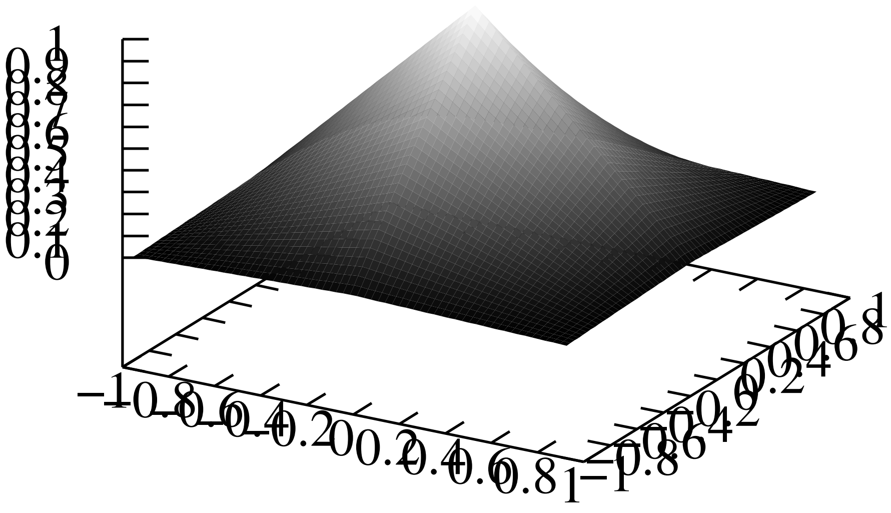


Wavelet Function

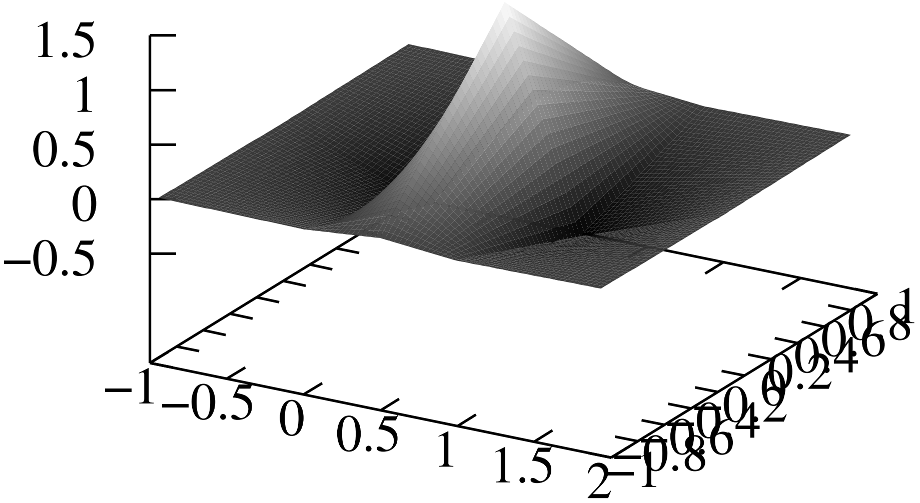


Wavelet Function

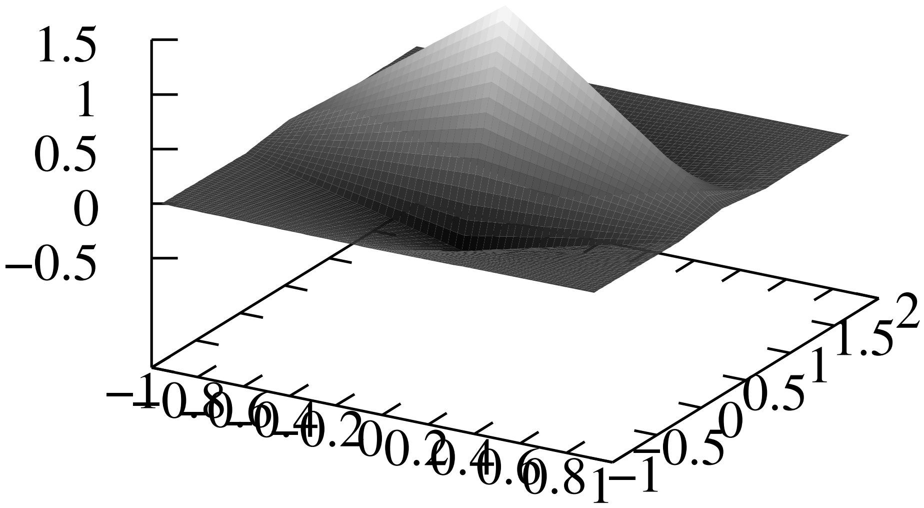
LeGall 5/3 Primal Scaling and Wavelet Functions



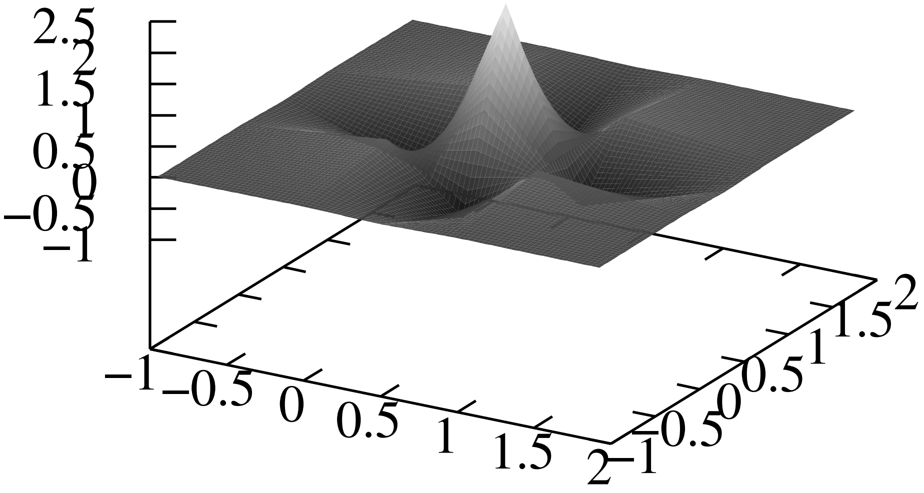
Scaling Function



Wavelet Function



Wavelet Function



Wavelet Function

Case Study: JPEG-2000 Image Compression Standard

REFER TO SLIDE PRESENTATION ON JPEG 2000.

Section 7.2

Signal Denoising

Signal Denoising

- noise often concentrated in particular frequency ranges (i.e., noise is often colored, not white)
- can be convenient to perform noise removal based on subband decomposition
- for example, if noise known to correspond to high frequencies, might attenuate subband signals associated with those frequencies

Section 7.3

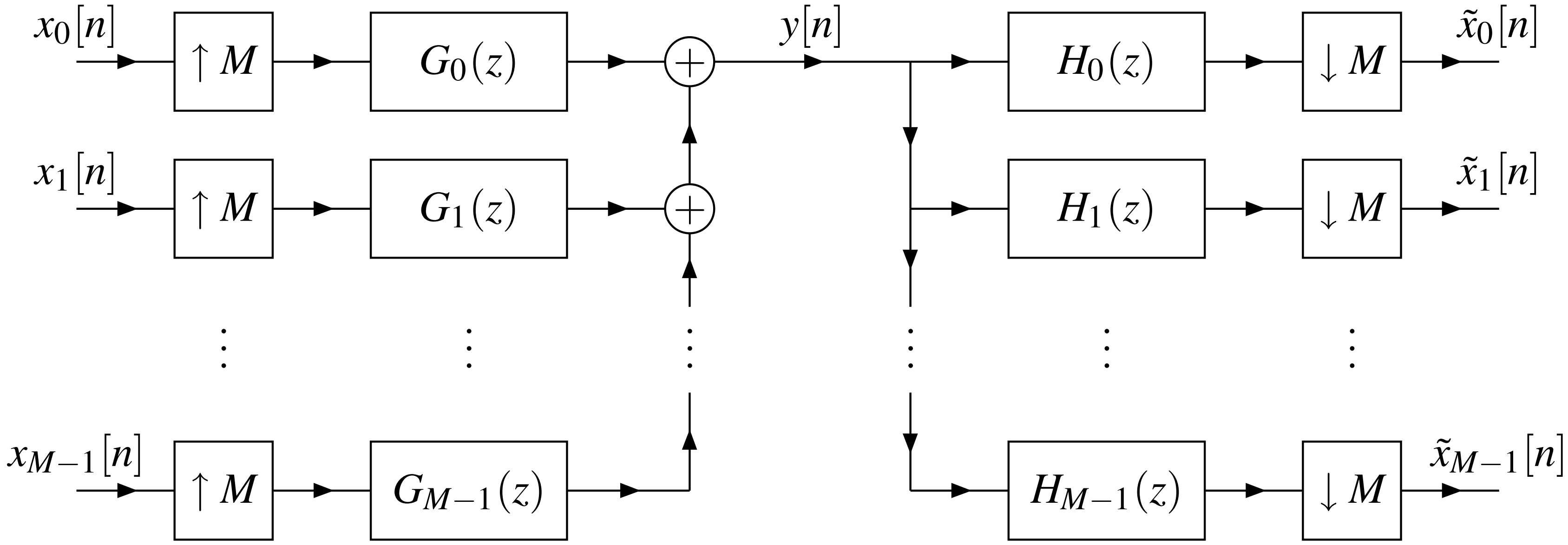
Transmultiplexers for Communications

Applications of Transmultiplexers

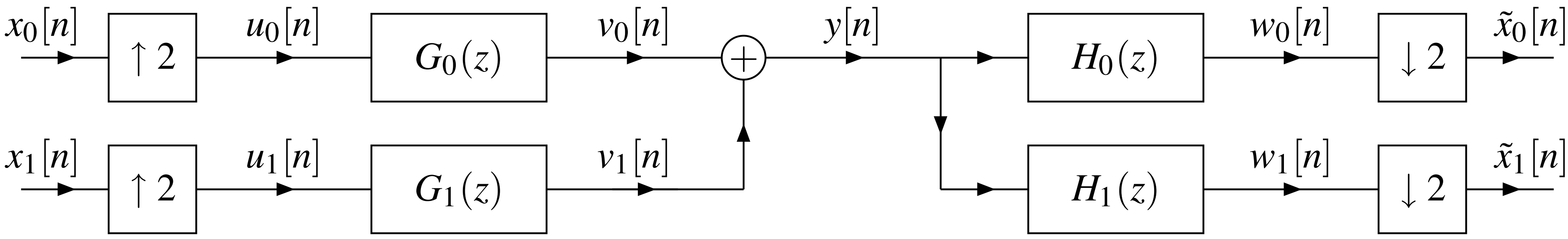
- useful in both single-user or multiple-user ***communication systems*** (e.g., single user utilizing multiple subchannels, multiple users each using one channel)
- used in frequency division multiple access (FDMA) systems; orthogonal frequency division multiplexing (OFDM) particularly popular; used in code division multiple access (CDMA) and time division multiple access (TDMA) systems
- for FDMA, analysis/synthesis filters chosen to be ***frequency selective*** (approximating ideal lowpass/bandpass/highpass filters)
- for TDMA, analysis/synthesis filters chosen to have simple ***one-tap*** impulses responses
- for CDMA, analysis/synthesis filters ***spread in both time and frequency***

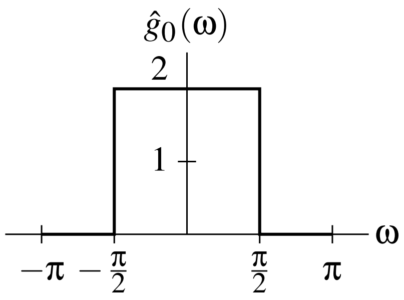
Multichannel Communication Systems

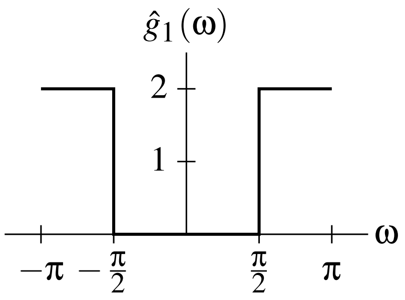
- in practice, transmultiplexer used to multiplex several signals over shared communication channel
- synthesis side (on left) multiplexes M signals $\{x_k\}_{k=0}^{M-1}$ onto single signal y
- analysis side (on right) demultiplexes y into M signals $\{\tilde{x}_k\}_{k=0}^{M-1}$

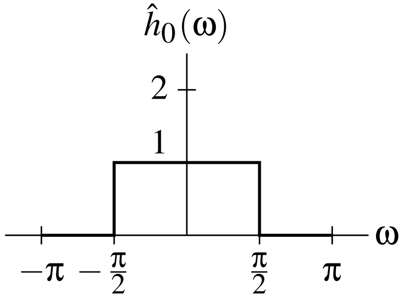


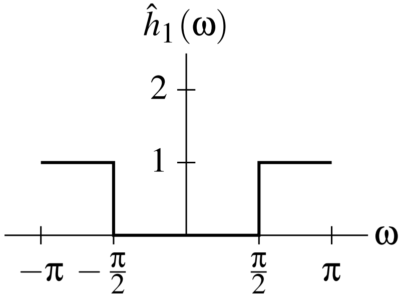
Two-Channel Transmultiplexer FDMA Example



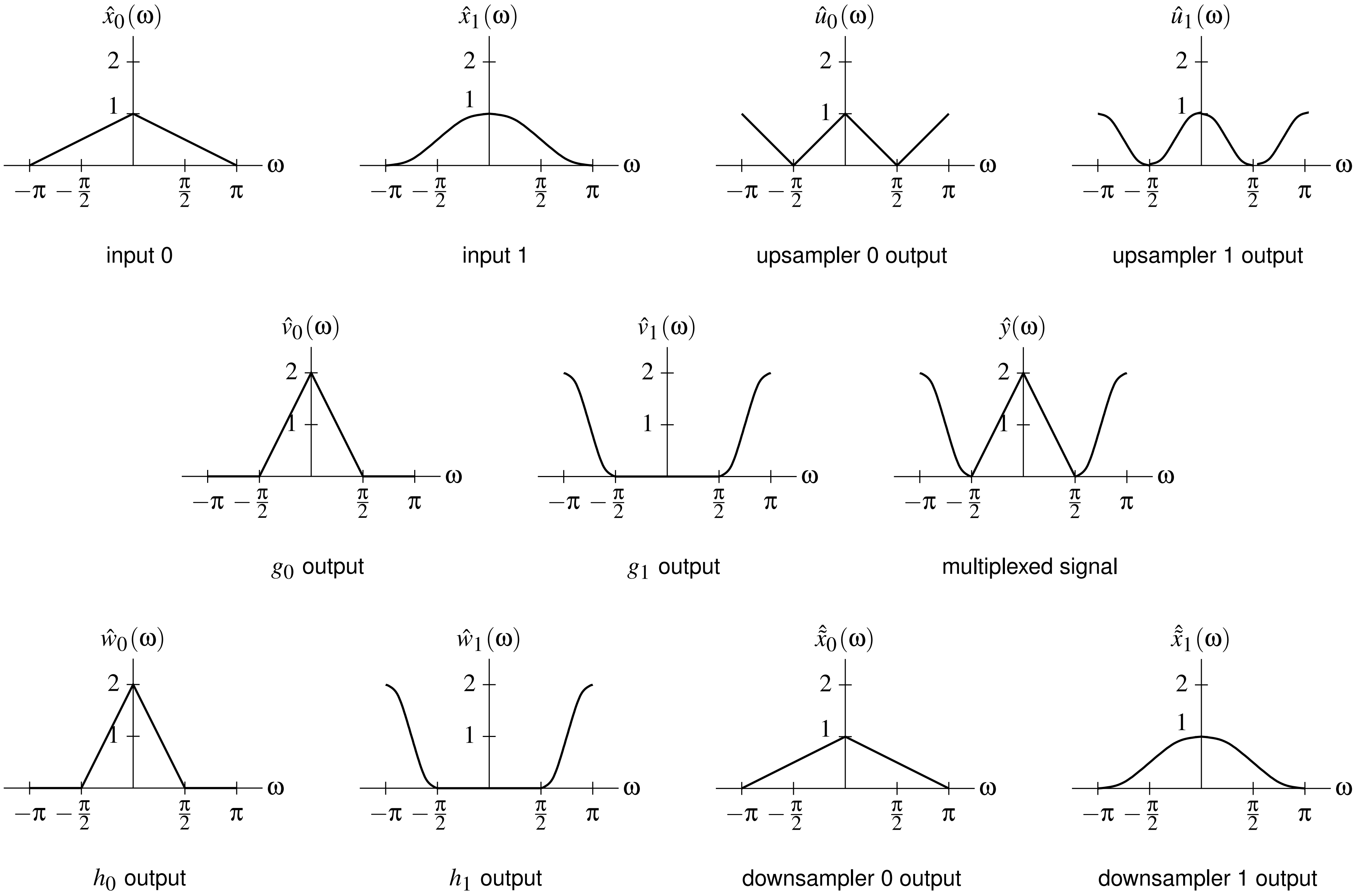








Two-Channel Transmultiplexer FDMA Example (Continued)



input 0

input 1

upsampler 0 output

upsampler 1 output

g_0 output

g_1 output

multiplexed signal

h_0 output

h_1 output

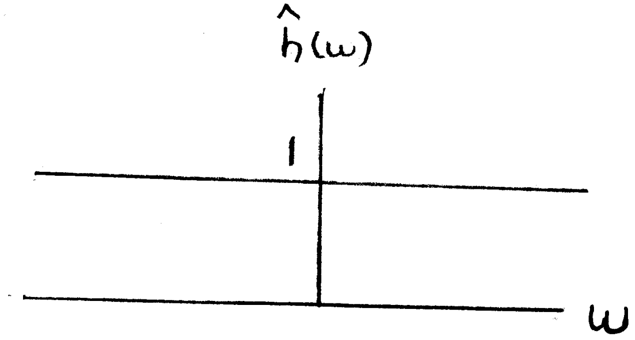
downsampler 0 output

downsampler 1 output

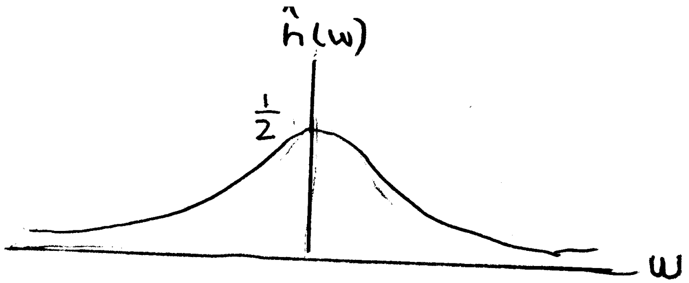
Multicarrier Modulation

- with **multicarrier modulation**, transmitted data ***split*** into several bit streams and used to modulate several carriers
- instead of transmitting ***single wideband*** signal over one channel, transmit ***set of narrowband*** signals
- used in various wireless LAN/MAN standards such as 802.11a, 802.15.3, 802.16a (WiMAX); used for high-speed data transmission over twisted pair channel of digital subscriber line (e.g., ADSL)
- multicarrier modulation has several advantages over classical single-carrier system
- adaptation of data rates of subchannels based on channel/noise characteristics (channel noise usually colored)
- can avoid transmitting in corrupted subchannels
- can transmit more important data in subchannels with high SNR
- facilitates ***more effective coding*** scheme to improve robustness to transmission errors
- channel effects can be ***more efficiently modelled*** (e.g., may be able to model transfer function of single narrowband channel as constant)

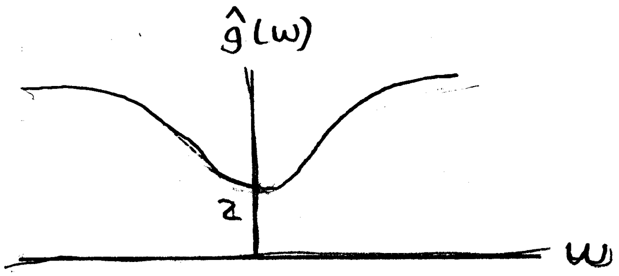
Multicarrier Modulation (Continued 1)



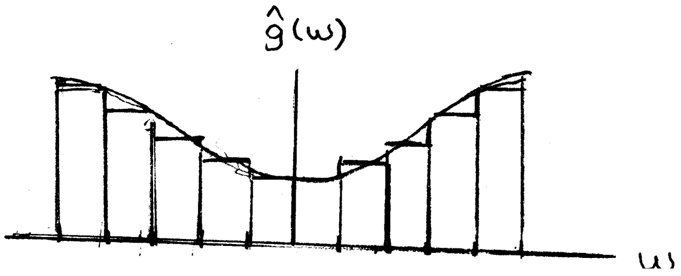
Ideal Channel



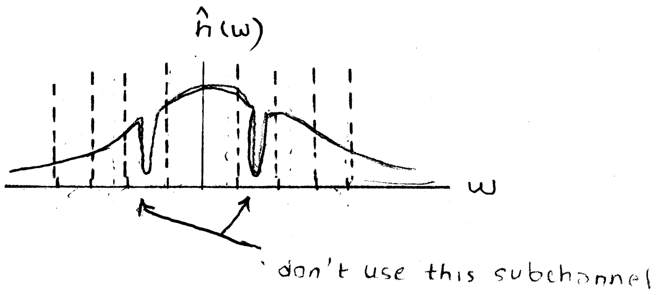
Real Channel



Equalizer



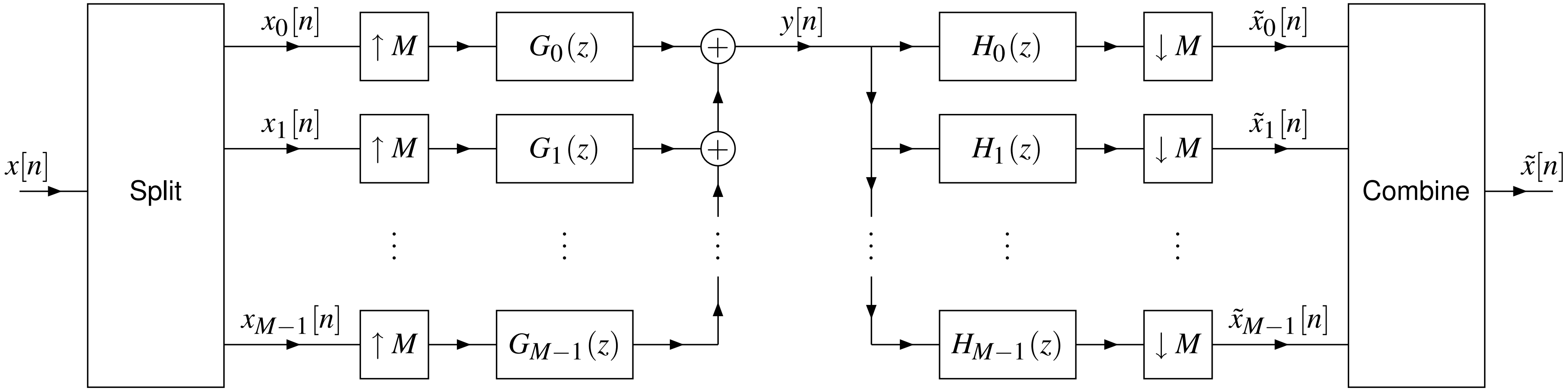
Per Channel Equalization



Real Channel

Multicarrier Modulation (Continued 2)

- input signal x split into M different data streams $\{x_k\}_{k=0}^{M-1}$
- all data streams are multiplexed together in single signal y which is sent over communication channel
- received signal y undergoes demultiplexing to obtain data streams $\{\tilde{x}_k\}_{k=0}^{M-1}$
- data streams combined together to form output signal $\tilde{x}$
- if system designed properly (i.e., PR property), $\tilde{x} = x$



Section 7.4

Other Applications

Adaptive Systems

- filter banks and multirate systems used in ***adaptive filtering*** applications (e.g., inverse filtering, room acoustics modelling, echo cancellation, equalization)
- adaptive filtering ***done in subbands***
- often ***shorter filters*** can be used
- some subbands may be ***ignored*** if contain little information

Analog Voice Privacy Systems

- digital voice data to be sent over analog channel with reasonable reproduction quality and some privacy
- in transmitter:
  1. split signal into M subbands (via filters and downsamplers)
  2. divide each subband signal into segments in time domain
  3. permute subbands and time segments and recombine to generate scrambled signal (with synthesis filter bank)
  4. D/A conversion
- in receiver:
  1. A/D conversion
  2. split signal into subbands (via filters and downsamplers)
  3. unshuffle subbands and time segments by applying inverse permutation (from that used in transmitter)
  4. recombine subband signals with synthesis side of filter bank

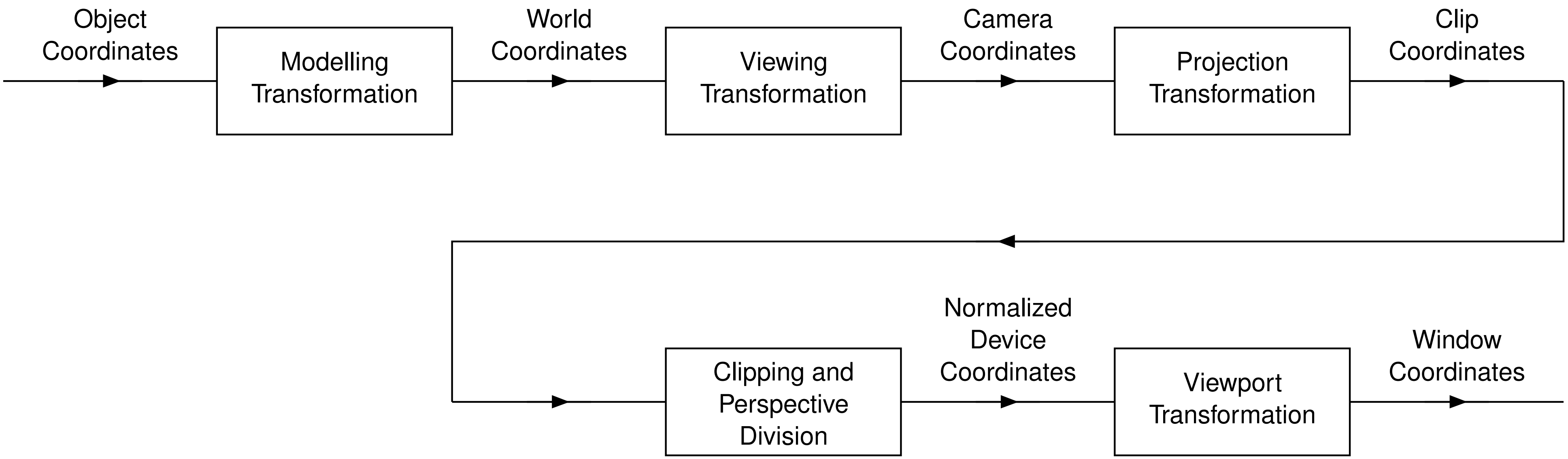
Part 8

Applications in Geometry Processing

Section 8.1

Computer Graphics

Transformations and Graphics Pipeline



- World coordinate system: The coordinate system for the scene in which various geometric objects are placed.
- Object coordinate system: The coordinate system for defining a particular geometric object, which may be different from the world coordinate system.
- Camera coordinate system: The coordinate system relative to which the scene is viewed.
- Window coordinate system: The coordinates used to address pixels in the graphics window.

Projection in Practice

- In a practical computer graphics systems, a projection is not actually performed.
- This is because the projection would discard depth information which is needed for clipping and hidden object removal.
- The projection matrix used in practice simply consists of transformation to position and orient the viewing plane appropriately along with, in the case of perspective projection, a warping.
- The actual projection itself, which would “flatten” the 3-D viewing volume onto the viewing plane is omitted.

Projection in Practice: Orthographic Case

- A commonly used orthographic projection maps the viewing volume $[l,r] \times [b,t] \times [n,f]$ to the cube $[-1,1] \times [-1,1] \times [-1,1]$ and is given by the matrix

$$P(l,r,t,b,n,f) = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & \frac{r+l}{l-r} \\ 0 & \frac{2}{t-b} & 0 & \frac{t+b}{b-t} \\ 0 & 0 & \frac{2}{n-f} & \frac{f+n}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The above matrix is the one used by the `glOrtho` function in OpenGL.

Projection in Practice: Perspective Case

- Suppose that the eye is at the origin looking in the negative z direction. We can specify a perspective projection by the parameters: θ, the field of view in the y direction; a, the aspect ratio (which determines the field of view in the x direction); n, the z coordinate of the near clipping plane; and f, the z coordinate of the far clipping plane.
- The matrix associated with the above transform is given by

$$P = \begin{bmatrix} \frac{c}{a} & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & \frac{f+n}{n-f} & \frac{2fn}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad \text{where } c = \cot\theta/2.$$

- The above matrix is the one used by the `gluPerspective` function in OpenGL.

Practical Use of Subdivision: Rendering

- In addition to the vertices of a mesh and their connectivity, to render a mesh, we need surface normals.
- In practice, given a control mesh, we would typically render the corresponding surface using a process like the following.
- Apply several iterations of subdivision to the control mesh in order to produce a refined mesh.
- Push the vertices in refined mesh to their limit positions using position masks.
- Using tangent masks, compute two tangent vectors at each point on the limit surface corresponding to a modified vertex position.
- From the two tangent vectors, compute a corresponding surface normal (via a vector cross product).
- Equipped with the vertex positions and normals from above, we then proceed to render the surface.
- We might, for example, use a rendering engine such as OpenGL in conjunction with a Blinn-Phong lighting model.

Part 9

Signal Processing Library (SPL)

Signal Processing Library (SPL)

- signal processing library (SPL) developed to provide basic multirate signal processing capabilities
- uses namespace `SPL`
- provides support for: one- and two-dimensional arrays; one- and two-dimensional sequences; downsampling, upsampling, convolution; polyphase decomposition/recomposition; lowpass, highpass, bandpass filter design;
- has other miscellaneous functionality (e.g., arcball, CGAL utilities)
- let `$SPL_TOPDIR` denote top level directory of SPL installation (which might be something like `/home/frodo/public/elec486/SPL`)
- header files in `$SPL_TOPDIR/include` directory; library files in `$SPL_TOPDIR/lib` directory; demo files in `$SPL_TOPDIR/share/SPL/demo` directory
- all header files in `SPL` directory; so to include SPL header file called `file.hpp` must use include directive like:

```
    #include <SPL/file.hpp>
```

- ***user manual available*** in both HTML and PDF formats

Header Files

Header File	Description
`Array1.hpp`	one-dimensional array class template (`Array1`)
`Array2.hpp`	two-dimensional array class template (`Array2`)
`Sequence1.hpp`	one-dimensional sequence class template (`Sequence1`)
`Sequence2.hpp`	two-dimensional sequence class template (`Sequence2`)
`filterDesign.hpp`	basic lowpass, highpass, and bandpass filter design
`Timer.hpp`	timer class (`Timer`)
`math.hpp`	math functions (e.g., `absVal`, `sqr`, `sinc`, `radToDeg`, `degToRad`)
`audioFile.hpp`	functions for loading/saving audio data in WAV format
`pnmCodec.hpp`	functions for loading/saving image data in PNM format

Header Files (Continued)

Header File	Description
`bitStream.hpp`	bit stream I/O classes
`arithCoder.hpp`	binary and M-ary arithmetic coder classes
`mCoder.hpp`	M-coder (binary) arithmetic coder classes
`version.hpp`	library version information

Array1 Class Template

- one-dimensional array of elements of arbitrary type
- uses reference counting and lazy copying
- Array1 declared as:

```
template <class T> class Array1;
```

- T: type of elements in array
- provides all of basic operations that one would expect of one-dimensional array class

Array1 Example

```
#include <iostream>
#include <SPL/Array1.hpp>
#include <SPL/audioFile.hpp>
#include <cmath>

typedef SPL::Array1<double> Array;

int main(int argc, char** argv)
{
    const int sampRate = 44100;
    const int numSamps = 5 * sampRate;
    const double sampPer = 1.0 / sampRate;
    const std::string outFile("sine.wav");
    Array samps(numSamps);
    for (int i = 0; i < numSamps; ++i) {
        samps(i) = 0.5 * (1.0 * sin(2.0 * M_PI * 440.0 *
          sampPer * i));
    }
    if (SPL::saveAudioFile(outFile, sampRate, samps)) {
        std::cerr << "cannot make audio file\n";
        return 1;
    }
    return 0;
}
```

Array2 Class Template

- two-dimensional array of elements of arbitrary type
- uses reference counting and lazy copying
- Array2 declared as:

```
template <class T> class Array2;
```

- T: type of elements in array
- provides all of basic operations that one would expect of two-dimensional array class

Sequence1 Class Template

- sequence defined on finite range of integers
- Sequence1 declared as:

```
    template <class T> class Sequence1;
```

- T: type of elements in sequence
- similar to one-dimensional array of size N, except that indices do not have to number starting at zero; generally indices number from L to $L+N-1$ inclusive; also, many additional operations provided
- some operations that involve more than one sequence may require particular relationship between domain of sequences (e.g., domains must be same)

Members Types

Member Type	Description
`ElemType`	type of element in sequence
`Iterator`	random-access iterator type
`ConstIterator`	const random-access iterator type

Member Functions

Construction, Destruction, and Assignment

Member Name	Description
constructor	construct
destructor	destroy
`operator=`	assign

Iterators

Member Name	Description
`begin`	get iterator referring to first element
`end`	get iterator referring to one-past last element

Member Functions

Compound Assignment

Member Name	Description
`operator+=`	add quantity to sequence
`operator-=`	subtract quantity from sequence
`operator*=`	multiply sequence by quantity
`operator/=`	divide sequence by quantity

Initializers and Accessors

Member Name	Description
`getStartInd`	get index of first element
`getEndInd`	get index of last element plus one
`getSize`	get number of elements
`operator()`	access element
`clear`	delete all elements
`fill`	set all elements to particular value

Non-Member Functions

Operators

Name	Description
`operator>>`	stream extractor (i.e., input)
`operator<<`	stream inserter (i.e., output)
`operator+`	addition
`operator-`	subtraction
`operator*`	multiplication
`operator/`	division

Non-Member Functions

Various Operations

Name	Description
`subsequence`	extract subsequence
`add`	add sequences (domains may differ)
`translate`	translate (i.e., shift) sequence
`upsample`	upsample sequence
`downsample`	downsample sequence
`convolve`	convolve two sequences
`polyphaseSplit`	compute polyphase decomposition
`polyphaseJoin`	compute polyphase recomposition

Filter Design

Name	Description
`lowpassFilter`	design lowpass filter
`highpassFilter`	design highpass filter
`bandpassFilter`	design bandpass filter

Sequence1 Example

```cpp
#include <iostream>
#include <SPL/Sequence1.hpp>

int main(int argc, char** argv)
{
    const double data[] = { 0.5, 0.5 };
    SPL::Sequence1<double> h(0, 2, &data[0]);
    SPL::Sequence1<double> x;
    if (!(std::cin >> x)) {
        return 1;
    }
    std::cout
      << "x * h:" << SPL::convolve(x, h,
        SPL::ConvolveMode::full) << "\n"
      << "2 x: " << 2.0 * x << "\n"
      << "x[n-1]: " << SPL::translate(x, 1) << "\n"
      << "down2 x: " << SPL::downsample(x, 2)
        << "\n"
      << "up2 x: " << SPL::upsample(x, 2) << "\n";

    return 0;
}
```

Sequence2 Class Template

- sequence defined on finite rectangular grid
- Sequence2 declared as:

```
    template <class T> class Sequence2;
```

- T: type of elements in sequence
- similar to two-dimensional array, except that indices do not have to number starting at zero; also, many additional operations provided
- some operations that involve more than one sequence may require particular relationship between domain of sequences (e.g., domains must be same)

Timer Class

- provides mechanism for making timing measurements
- can start and stop timers and query elapsed time measured by timer

Member Functions

Constructors, Destructor, and Assignment

Name	Description
constructor	construct
destructor	destroy
`operator=`	assign

Timing Functions

Name	Description
`start`	start timer
`stop`	stop timer
`get`	get elapsed time

`Timer` Example

```cpp
#include <iostream>
#include <cmath>
#include <SPL/Timer.hpp>

int main(int argc, char** argv)
{
    SPL::Timer timer;

    timer.start(); // Start the timer.

    // Perform some computation.
    double sum = 0.0;
    for (int i = 0; i < 100000; ++i)
        sum += exp(i / 50000.0) *
          cos(2.0 * M_PI / 100.0 * i);
    std::cout << "sum: " << sum << "\n";

    timer.stop(); // Stop the timer.

    // Get the elapsed time.
    double t = timer.get();
    std::cout << "elapsed time: " << t << "\n";

    return 0;
}
```

Part 10

Computational Geometry Algorithms Library (CGAL)

Computational Geometry Algorithms Library (CGAL)

- very powerful open-source C++ library for geometric computation
- used by many commercial organizations such as British Telecom, Boeing, France Telecom, GE Health Care, The MathWorks
- very well documented (extensive manual, more than 4000 pages)
- provides data types for representing various geometric objects (e.g., triangle/quadrilateral/polygon meshes) and algorithms for manipulating these data types
- provides support for polygon meshes (e.g., triangle/quadrilateral meshes)
- can read/write polygon mesh data in various common formats
- built-in support for several subdivision schemes
- by using CGAL, can greatly simplify amount of effort required to implement methods using subdivision surfaces or wavelet transforms for triangle meshes
- available for Microsoft Windows and Unix/Linux platforms
- some Linux distributions already have packages for CGAL (e.g., Fedora packages: `CGAL`, `CGAL-devel`, `CGAL-demos-source`)
- home page: `http://www.cgal.org`

CGAL (Continued)

- most relevant material (in CGAL 3.6.1 manual):
  - Chapter 25 titled “3D Polyhedral Surfaces”
  - Chapter 50 titled “3D Surface Subdivision Methods”
- `Polyhedron_3` class for representing polyhedrons (employs half-edge data structure)
- several functions for implementing subdivision schemes:
  - `Loop_subdivision` (Loop subdivision)
  - `Sqrt3_subdivision` (Kobbelt $\sqrt{3}$ subdivision)
  - `PTQ` (primal triangle quadrisection with user-defined geometric refinement rule)
  - `Sqrt3` ($\sqrt{3}$ topologic refinement with user-defined geometric refinement rule)
- found bugs in Loop and Kobbelt $\sqrt{3}$ subdivision implementations in CGAL 3.5.1:
  - Loop subdivision handles extraordinary vertices incorrectly
  - Kobbelt $\sqrt{3}$ subdivision updates old vertices incorrectly

CGAL Information Resources

- CGAL Project Home Page:

  `http://www.cgal.org`
- CGAL Manuals (various versions in PDF and HTML formats):

  `http://www.cgal.org/Manual`
- CGAL Manual (latest version in HTML format):

  `http://www.cgal.org/Manual/latest`
- CGAL Discussion List:

  `https://lists-sop.inria.fr/sympa/arc/cgal-discuss`

Handles, Iterators, and Circulators

- **handle**: type that references elements stored in some data structure (i.e., type can be dereferenced to obtain access to element)
- for data structure storing elements of type `T`, examples of handles include: simple pointer `T*`, iterator with value type `T`
- example of handles: types used to access vertices, facets, halfedges of polygon mesh
- ***iterators*** are very useful, but intended for use with linear sequences of elements (i.e., sequences with well-defined first and last element)
- often want iterator-like functionality for circular sequences of elements
- **circulator**: type that allows iteration over elements in circular sequence of elements
- example of circulator: type to allow iteration over all 1-ring neighbours of vertex in polygon mesh
- circulators come in const and mutable (i.e., non-const) versions (mutable circulators can be used to modify referenced element, while const circulator cannot)

Section 10.1

Geometry Kernels

Real Number Types

- `float`: single-precision floating point type
- `double`: double-precision floating point type
- `Interval_nt`: interval-arithmetic type
- `MP_Float`: arbitrary-precision floating-point type

MP_Float Class

- MP_Float
- arbitrary-precision floating-point type
- additions, subtractions, and multiplications computed exactly
- does not provide division or square root (which is not typically problematic as division rarely needed and square root almost never used by any sane person in geometric computation)
- no roundoff error
- no overflow error unless astronomically large numbers involved (arbitrary length mantissa; integral-valued double exponent can overflow, but extremely unlikely)
- very slow, can require considerable memory (unbounded)

`Interval_nt` Class

- declared as: `template <bool M = true> Interval_nt<M>`
- `M` indicates if safe rounding mode enabled
- if safe rounding mode enabled, rounding mode always restored to round towards zero (required by C++); must be careful if safe rounding mode not used
- when safe rounding mode not used, faster but need to worry about things like compiler options like `-frounding-math`
- `typedef Interval_nt<false> Interval_nt_advanced;` (i.e., `Interval_nt_advanced` is `Interval_nt` with safe rounding mode disabled)
- interval-arithmetic number type (internally uses floating-point type)
- represents interval $[a,b]$
- every arithmetic operation performed twice, once while rounding towards $-\infty$ to produce result a' and once while rounding towards $+\infty$ to produce result b'
- true answer must lie on interval $[a',b']$
- approximately twice of time cost of built-in floating-point type

Geometry Kernels

- represent geometric objects (e.g., point, line, line segment, ray, plane, triangle, circle,)
- points in 2 or 3 dimensions
- provide operations on geometric objects (e.g., intersection, composition)
- allow certain conditions to be tested involving geometric objects (e.g., collinear, coplanar, equality)

Point Representation

- Cartesian kernels: coordinates represented in Cartesian form
- homogeneous kernels: coordinates represented in homogeneous form

Simple_cartesian and Cartesian Classes

- geometry kernel that represents coordinates in Cartesian form
- declaration:

```
template <class F> Simple_cartesian<F>
```

- declaration:

```
template <class F> Cartesian<F>
```

- F field number type (used to represent coordinates)
- F often chosen as double
- Cartesian is reference counted version of Simple_cartesian, which allows more efficient copying of objects
- Cartesian probably preferred if frequent copying occurs

`Simple_homogeneous` and `Homogeneous` Classes

- geometry kernel that represents coordinates in homogeneous form
- declaration:

```
template <class R> Simple_homogeneous<R>
```

- declaration:

```
template <class R> Homogeneous<R>
```

- `R` ring number type used for representing numerator and denominator of rational coordinates
- `Homogeneous` is reference counted version of `Simple_homogeneous`, which allows more efficient copying of objects
- `Homogeneous` probably preferred if frequent copying occurs

Constructions

- produces new geometric object from other objects
- result is not one of a small number of enumerable values
- result is numerical (e.g., involves real numbers)
- create line segment from two points
- create triangle from three points
- create plane from three (non-coplanar) points
- create circle from three (non-collinear) points
- find intersection of line and plane
- exact construction: any newly created geometric objects resulting from construction are exactly represented (i.e., no roundoff/overflow error)
- inexact construction: newly created geometric objects are not guaranteed to be exactly represented (e.g., due to roundoff error)
- extremely important to be aware of whether kernel being used provides exact constructions; affects how you write code!!!

Predicates

- does not involve any newly computed numerical data
- result is one of very small set of values, such as boolean or enumerated type
- typically used to make decisions (i.e., affect control flow)
- are three points collinear (true or false)
- are four points coplanar (true or false)
- what is position of point relative to oriented line (left of, right of, or on)
- what is position of point relative to oriented circle (inside, outside, or on)
- exact predicate: result of test is guaranteed to be correct (i.e., result determined as if by exact computation)
- inexact predicate: result of test may be incorrect (e.g., due to roundoff/overflow error)
- extremely important to be aware of whether kernel being used provides exact predicates; affects how you write code!!!

Kernel Selection

- coordinate representation
- exact or inexact constructions
- exact or inexact predicates
- in practice, almost always require exact predicates
- if code well designed, need for exact constructions can usually be avoided
- for `T` chosen as any numeric type that has roundoff/overflow error (e.g., `float`, `double`, `long double`), the following kernels do not provide exact constructions or exact predicates:

```
Simple_cartesian<T>
Cartesian<T>
Simple_homogeneous<T>
Homogeneous<T>
```

`Filtered_kernel` Class

- class to convert kernel with inexact predicates into one with exact predicates
- declared as:

```
template <class K> Filtered_kernel<K>
```

- `K` is kernel from which to make filtered kernel
- predicates of `K` replaced by predicates using numeric type `Interval_nt`
- if interval arithmetic can yield reliable answer, result used
- otherwise, exception thrown and caught by class and predicate using `MP_Float` used
- for exact predicates with `Simple_cartesian<double>`, use: `Filtered_kernel<Simple_cartesian<double>>` or equivalently `Exact_predicates_inexact_constructions_kernel`
- `Exact_predicates_inexact_constructions_kernel` very commonly used

Writing Custom Exact Predicates

- exact predicate cannot at any point rely on a computation that is not exact
- no floating point arithmetic (since it has roundoff error)
- no integer arithmetic that might overflow
- no inexact constructions
- no inexact predicates
- `Filtered_predicate` may be helpful

Filtered_predicate Class

- adapter for predicate functors for producing efficient exact predicates
- declared as:

```
template <class EP, class FP, class CE, class CF>
Filtered_predicate<EP, FP, CE, CF>
```

- EP is exact predicate (typically uses arbitrary-precision type such as MP_Float)
- FP is filtering predicate (typically uses interval-arithmetic type like Interval_nt)
- CE and CF are function objects for converting arguments of unfiltered predicate to types used by exact and filtering predicates
- must be careful about operation used in unfiltered predicate being plugged into Filtered_kernel
- for kernel ring number type RT, can safely use addition, subtraction, multiplication
- can also safely use sign

Execution of Filtered Predicate

- execution of code for filtered predicate functor proceeds as follows:
  1. invoke unfiltered (i.e., original) predicate functor for numeric type `CGAL::Interval_nt<false>`
     if any operation on interval arithmetic type yields uncertain result (e.g., `CGAL::sign`), exception is thrown, with thrown exception being caught by filtered predicate functor
  2. if no exception thrown (so that unfiltered functor returns normally), return return value of unfiltered functor (and we are done); otherwise, continue
  3. invoke unfiltered predicate functor for numeric type `CGAL::MP_Float`
  4. return return value of unfiltered functor

Filtered Predicate Example

```
#include <CGAL/Cartesian.h>
#include <CGAL/MP_Float.h>
#include <CGAL/Interval_nt.h>
#include <CGAL/Filtered_predicate.h>
#include <CGAL/Cartesian_converter.h>

template <class K>
struct Test_orientation_2 {
    typedef typename K::RT RT;
    typedef typename K::Point_2 Point_2;
    typedef typename K::Orientation result_type;
    result_type operator()(const Point_2& p, const Point_2& q,
      const Point_2& r) const {
        RT prx = p.x() - r.x();
        RT pry = p.y() - r.y();
        RT qrx = q.x() - r.x();
        RT qry = q.y() - r.y();
        return CGAL::sign(prx * qry - qrx * pry);
    }
};

typedef CGAL::Cartesian<double> Kernel;
typedef CGAL::Cartesian<CGAL::Interval_nt<false>> Ia_kernel;
typedef CGAL::Cartesian<CGAL::MP_Float> Exact_kernel;
typedef CGAL::Filtered_predicate<
  Test_orientation_2<Exact_kernel>,
  Test_orientation_2<Ia_kernel>,
  CGAL::Cartesian_converter<Kernel, Exact_kernel>,
  CGAL::Cartesian_converter<Kernel, Ia_kernel>
  > Test_orientation;

int main() {
    double big = 1e50;
    Kernel::Point_2 p(0.0, 0.0), q(1.0, 1.0), r(2.0 * big, 2.0 * big);
    Test_orientation orientation;
    std::cout << orientation(p, q, r) << "\n";
}
```

Filtered Predicate Example (Continued)

- for example on previous slide, execution of filtered predicate functor proceeds as follows:
  1. invoke `Test_orientation_2<Cartesian<CGAL::Interval_nt<false>>>` functor with points $([0,0],[0,0])$, $([1,1],[1,1])$, $([2\cdot 10^{50}, 2\cdot 10^{50}],[2\cdot 10^{50}, 2\cdot 10^{50}])$
  2. `CGAL::sign` called for $[-1.55414\cdot 10^{85}, 1.55414\cdot 10^{85}]$, which results in exception being thrown
  3. exception caught by filtered predicate code
  4. invoke `Test_orientation_2<Cartesian<CGAL::MP_Float>>` functor with points $(0,0)$, $(1,1)$, $(2\cdot 10^{50}, 2\cdot 10^{50})$
  5. `CGAL::sign` called for 0, resulting in return value of 0
  6. filtered predicate returns 0

- critically important that `RT` used for all arithmetic operations and not `double` (or `float`); otherwise, arithmetic computation done using wrong numeric type, which will prevent predicate from being correct (i.e., exact)

Section 10.2

Polygon Meshes

Polyhedron_3 Class

- represents a polyhedral surface (i.e., polygon mesh), which consists of vertices, edges, and facets and incidence relationship amongst them
- each edge represented by pair of halfedges
- declaration for Polyhedron_3 class:

```
template <class Kernel,
  class PolyhedronItems = CGAL::Polyhedron_items_3,
  template <class T, class I>
    class HalfedgeDS = CGAL::HalfedgeDS_default,
  class Alloc = CGAL_ALLOCATOR(int)>
class Polyhedron_3;
```

- Kernel is geometric kernel, which specifies such things as how points are represented and provides basic geometric operations/predicates (CGAL::Cartesian<double> most likely to be used in this course)
- PolyhedronItems_3 specifies data types for representing vertices and facets (in most, but not all cases, default will suffice)
- other template parameters for Polyhedron_3 beyond scope of course

Polyhedron_3 Type Members

Basic Types

Type	Description
Vertex	vertex type
Halfedge	halfedge type
Facet	facet type
Point_3	point type (for vertices)

Handles

Type	Description
Vertex_const_handle	const handle to vertex
Vertex_handle	handle to vertex
Halfedge_const_handle	const handle to halfedge
Halfedge_handle	handle to halfedge
Facet_const_handle	const handle to facet
Facet_handle	handle to facet

`Polyhedron_3` Type Members (Continued 1)

Iterators

Type	Description
`Vertex_const_iterator`	const iterator over all vertices
`Vertex_iterator`	iterator over all vertices
`Halfedge_const_iterator`	const iterator over all halfedges
`Halfedge_iterator`	iterator over all halfedges
`Facet_const_iterator`	const iterator over all facets
`Facet_iterator`	iterator over all facets
`Edge_const_iterator`	const iterator over all edges (every other halfedge)
`Edge_iterator`	iterator over all edges (every other halfedge)

Polyhedron_3 Type Members (Continued 2)

Circulators

Type	Description
Halfedge_around_vertex_const_circulator	const circulator of halfedges around vertex (CW)
Halfedge_around_vertex_circulator	circulator of halfedges around vertex (CW)
Halfedge_around_facet_const_circulator	const circulator of halfedges around facet (CCW)
Halfedge_around_facet_circulator	circulator of halfedges around facet (CCW)

Polyhedron_3 Function Members

Size

Name	Description
size_of_vertices	get number of vertices
size_of_halfedges	get number of halfedges
size_of_facets	get number of facets

Iterators

Name	Description
vertices_begin	iterator over all vertices
vertices_end	past-the-end iterator
halfedges_begin	iterator over all halfedges
halfedges_end	past-the-end iterator
facets_begin	iterator over all facets
facets_end	past-the-end iterator
edges_begin	iterator over all edges
edges_end	past-the-end iterator

Polyhedron_3 Function Members (Continued 1)

Combinatorial Predicates

Name	Description
`is_closed`	true if no border edges (no boundary)
`is_pure_triangle`	true if all facets are triangles
`is_pure_quad`	true if all facets are quadrilaterals

Polyhedron_3 Function Members (Continued 2)

Border Halfedges

Name	Description
normalized_border_is_valid	true if border is normalized
normalize_border	sort halfedges such that non-border edges precede border edges (i.e., normalize border)
size_of_border_halfedges	get number of border halfedges (border must be normalized)
size_of_border_edges	get number of border edges (border must be normalized)
border_halfedges_begin	halfedge iterator starting with border edges (border must be normalized)
border_edges_begin	edge iterator starting with border edges (border must be normalized)

Polyhedron_3::Facet

- represents facet (i.e., face) in polyhedral surface
- optional information: plane equation, reference to incident halfedge that points to facet
- circulator either forward or bidirectional

Facet Function Members

Name	Description
halfedge	get incident halfedge that points to facet
facet_begin	get circulator of halfedges around facet (CCW)
facet_degree	get degree of facet (i.e., number of edges on boundary of facet)
is_triangle	true if facet is triangle
is_quad	true if facet is quadrilateral

Polyhedron_3::Vertex

- represents vertex in polyhedral surface
- optional information: point, reference to incident halfedge that points to vertex
- circulator is either forward or bidirectional

Vertex Function Members

Queries

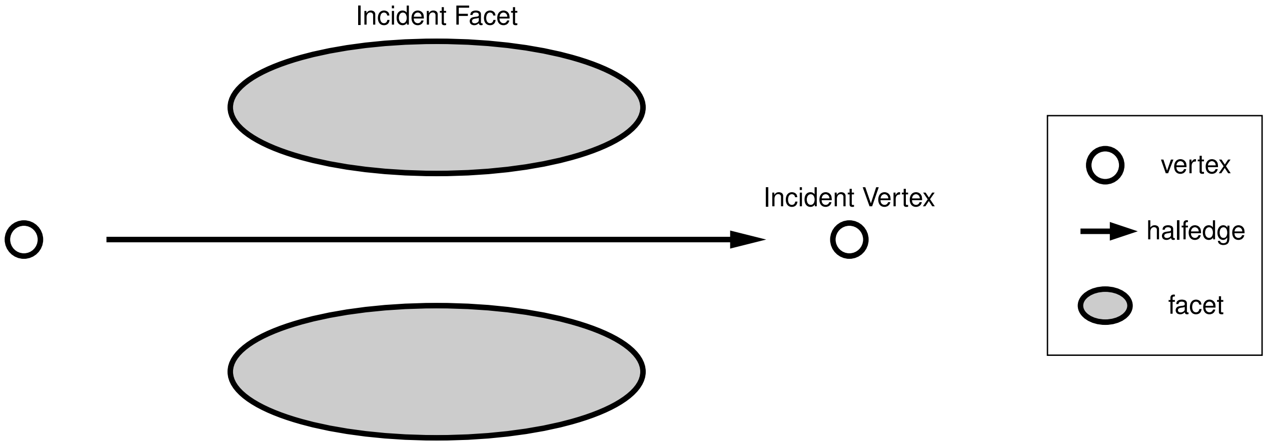
Name	Description
vertex_begin	circulator of halfedges around vertex (CW)
vertex_degree	get valence of vertex
is_bivalent	true if vertex has valence two
is_trivalent	true if vertex has valence three

Queries (Optional)

Name	Description
point	get point associated with vertex
halfedge	get incident halfedge that points to vertex

Polyhedron_3::Halfedge

- each halfedge directly associated with one vertex and one facet, referred to as incident vertex and incident facet, respectively
- **incident vertex** is vertex at ***terminal end*** of halfedge
- **incident facet** is facet to ***left*** of halfedge
- halfedge contains handle (i.e., pointer) for next halfedge around its left facet in CCW direction and handle (i.e., pointer) for its opposite halfedge
- this allows for efficient iteration around halfedges of facet in CCW direction and around halfedges of vertex in CW direction



`Halfedge` Function Members

Adjacency Queries

Name	Description
`opposite`	get opposite halfedge
`next`	next halfedge around facet (CCW)
`prev`	previous halfedge around facet (CCW)
`next_on_vertex`	next halfedge around vertex (CW)
`prev_on_vertex`	previous halfedge around vertex (CW)

Circulators

Name	Description
`vertex_begin`	circulator of halfedges around vertex (CW)
`facet_begin`	circulator of halfedges around facet (CCW)

Halfedge Function Members

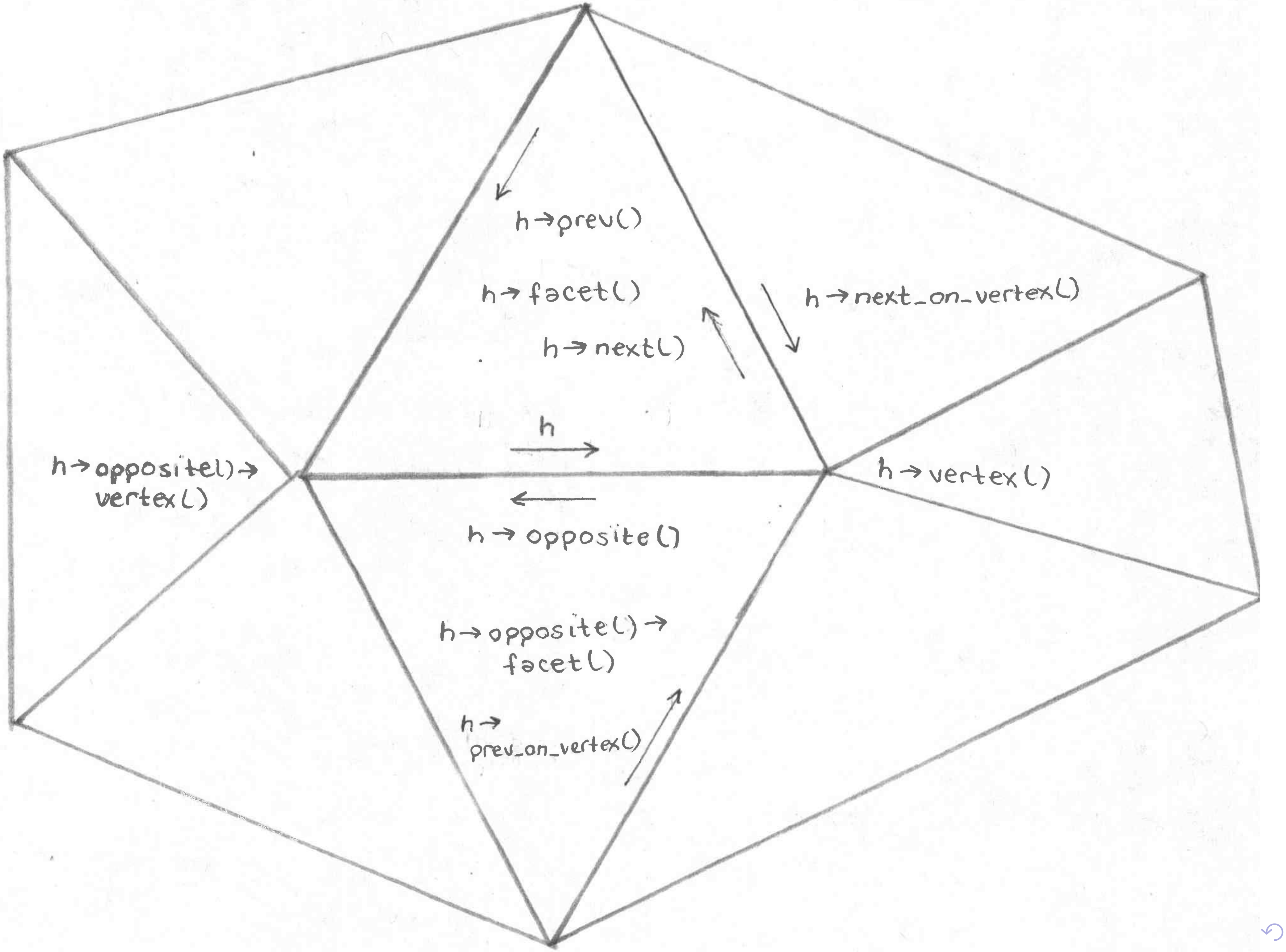
Queries

Name	Description
`is_border`	true if border halfedge
`vertex_degree`	get valence of incident vertex
`is_bivalent`	true if incident vertex has valence two
`is_trivalent`	true if incident vertex has valence three
`facet_degree`	get degree of incident facet
`is_triangle`	true if incident facet is triangle
`is_quad`	true if incident facet is quadrilateral

Incident Vertex/Facet

Name	Description
`vertex`	get handle for incident vertex of halfedge
`facet`	get handle for incident facet of halfedge

Adjacency Example



Polyhedron_3 I/O

- operator<< and operator>> are overloaded for I/O
- read and write polygon mesh data in OFF format

CGAL Gotchas

- be careful about operations on `Polyhedron_3` that invalidate handles, iterators, or circulators
- be careful about orientations (CCW versus CW)

CGAL Example Programs

1. ***Mesh Generation* (`meshMake.cpp`).** Create a mesh corresponding to a tetrahedron. Output the resulting mesh in Object File Format (OFF).
2. ***Mesh Information* (`meshInfo.cpp`).** Input a mesh in Object File Format (OFF). Calculate and print some information about the mesh (e.g., triangle or quad mesh, the number of vertices, edges, faces, halfedges, the minimum, maximum, and average vertex valence).
3. ***Mesh Subdivision* (`meshSubdivide.cpp`).** Input a mesh in Object File Format (OFF). Subdivide the mesh using the specified method and number of levels (where the Loop and Kobbelt $\sqrt{3}$ methods are supported). Output the refined mesh in Object File Format (OFF).

Part 11

Open Graphics Library (OpenGL)

Open Graphics Library (OpenGL)

- application programming interface (API) for high-performance high-quality 2-D and 3-D graphics
- most widely adopted 2-D and 3-D graphics API in industry
- bindings for numerous programming languages (i.e., C, Java, Fortran)
- window-system and operating-system independent
- available on all mainstream systems (e.g., Microsoft Windows, Linux/Unix, Mac OS X)
- vendor-neutral, controlled by independent consortium with many organizations as members (including, not surprisingly, companies like Intel, NVIDIA, and AMD)
- official web site: `http://www.opengl.org`
- OpenGL ES for embedded systems (e.g., mobile phones, game consoles, personal navigation devices, personal media players, automotive systems, settop boxes)

OpenGL (Continued)

- comprised of several hundred functions
- geometric primitives consist of points, lines, polygons, images, and bitmaps
- arrange geometric primitives in 3-D space and select desired vantage point for viewing composed scene
- calculate colors of objects (e.g., by explicit assignment, lighting, texture mapping, or combination thereof)
- convert mathematical description of objects to pixels on screen (i.e., rasterization)
- can eliminate hidden parts of objects (depth buffering), perform antialiasing, and so on
- only concerned with rendering
- no mechanism provided for creating windows (which is window-system dependent)
- no mechanism for obtaining user input (e.g., via mouse or keyboard)
- another library must be used in conjunction with OpenGL in order to manage windows, user input, and so on

Section 11.1

OpenGL Utility Toolkit (GLUT)

OpenGL Utility Toolkit (GLUT)

- simple windowing API for OpenGL
- intended to be used with small to medium sized OpenGL programs
- language binding for C
- window-system independent
- supports most mainstream operating systems (Microsoft Windows, Linux/Unix)
- provides window management functionality (e.g., creating/destroying windows, displaying/resizing windows, and querying/setting window attributes)
- allows for user input (e.g., via keyboard, mouse)
- routines for drawing common wireframe/solid 3-D objects such as sphere, torus, and well-known teapot model
- register callback functions to handle various types of events (e.g., display, resize, keyboard, special keyboard, mouse, timer, idle) and then loop processing events
- open-source implementation of GLUT called Freeglut is available from `http://sourceforge.net/projects/freeglut`

GLUT

- first, we focus on GLUT library

Event-Driven Model

- event-driven model: flow of program determined by events (e.g., mouse clicks, key presses)
- application making use of event-driven model performs some initialization and then enters an event-processing loop for duration of execution
- each iteration of event-processing loop does following:
    1. wait for event
    2. process event
- many libraries for building graphical user interfaces (GUIs) employ event-driven model
- GLUT uses event-driven model

Structure of GLUT Application

1. initialize GLUT library by calling `glutInit`
2. set display mode (via `glutInitDisplay`)
3. perform any additional initialization such as:
   - create windows (via `glutCreateWindow`)
   - register callback functions for handling various types of events (e.g., via `glutDisplayFunc`, `glutReshapeFunc`, `glutKeyboardFunc`)
   - setup initial OpenGL state (e.g., depth buffering, shading, lighting, clear color)
4. enter main event-processing loop by calling `glutMainLoop` [Note that `glutMainLoop` never returns.]

GLUT Header Files

- OpenGL and GLUT header files in directory `GL`
- to use GLUT, need to include `glut.h`

```
#include <GL/glut.h>
```

- header file `glut.h` also includes all necessary OpenGL header files (e.g., `gl.h`, `glu.h`, `glext.h`)

Event Types

Event Type	Description
display	window contents needs to be displayed
overlay display	overlay plane contents needs to be displayed
reshape	window has been resized
keyboard	key has been pressed
mouse	mouse button has been pressed or released
motion	mouse moved within window while one or more buttons pressed
passive motion	mouse moved within window while no buttons pressed
visibility	visibility of window has changed (covered versus uncovered)
entry	mouse has left or entered window
special keyboard	special key has been pressed (e.g., arrow keys, function keys)

Event Types (Continued)

Event Type	Description
spaceball motion	spaceball translation has occurred
spaceball rotate	spaceball rotation has occurred
button box	button box activity has occurred
dials	dial activity has occurred
tablet motion	tablet motion has occurred
tablet button	table button has been pressed or released
menu status	menu status change
idle	no event activity has occurred
timer	timer has expired

Functions

Initialization

Function	Description
`glutInit`	initialize GLUT library
`glutInitWindowSize`	set initial window size for `glutCreateWindow`
`glutInitWindowPosition`	set initial window position for `glutCreateWindow`
`glutInitDisplayMode`	set initial display mode

Beginning Event Processing

Function	Description
`glutMainLoop`	enter GLUT event-processing loop

Functions (Continued 1)

Window Management

Function	Description
`glutCreateWindow`	create top-level window
`glutPostRedisplay`	mark current window as needing to be redisplayed
`glutSwapBuffers`	swaps buffers of current window if double buffered (flushes graphics output via `glFlush`)

Callback Registration

Function	Description
`glutDisplayFunc`	sets display callback for current window
`glutReshapeFunc`	sets reshape callback for current window
`glutKeyboardFunc`	sets keyboard callback for current window
`glutSpecialFunc`	sets special keyboard callback for current window
`glutTimerFunc`	registers timer callback to be triggered in specified number of milliseconds

Functions (Continued 2)

State Retrieval

Function	Description
`glutGet`	retrieves simple GLUT state (e.g., size or position of current window)
`glutGetModifiers`	retrieve modifier key state when certain callbacks generated (i.e., state of shift, control, and alt keys)

Geometric Object Rendering

Function	Description
`glutSolidSphere`	render solid sphere
`glutWireSphere`	render wireframe sphere
`glutSolidCube`	render solid cube
`glutWireCube`	render wireframe cube
`glutSolidTorus`	render solid torus
`glutWireTorus`	render wireframe torus

Minimal GLUT Application: Source Code

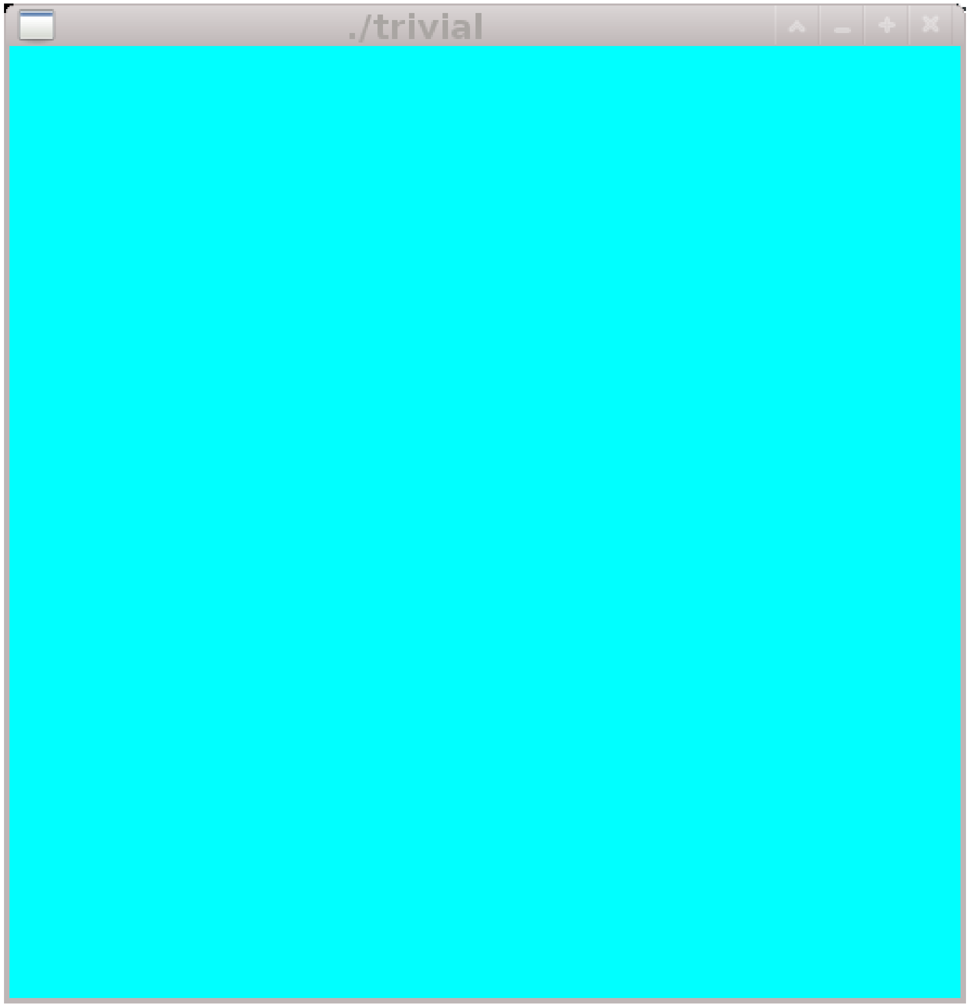
```
// Create a window that is cleared to a particular color
// when drawn.

#include <GL/glut.h>

void display() {
    glClearColor(0.0, 1.0, 1.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glutSwapBuffers();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(512, 512);
    glutCreateWindow(argv[0]);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Minimal GLUT Application: Graphics Output



Almost Minimal GLUT Application

```
// draw a light green square

#include <GL/glut.h>

void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glBegin(GL_QUADS);
    glColor3f(0.5, 1.0, 0.5);
    glVertex3f(-0.5, -0.5, 0.0);
    glVertex3f( 0.5, -0.5, 0.0);
    glVertex3f( 0.5,  0.5, 0.0);
    glVertex3f(-0.5,  0.5, 0.0);
    glEnd();
    glutSwapBuffers();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(512, 512);
    glutCreateWindow(argv[0]);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Other Useful GLUT References

- course web page has copy of GLUT manual in PDF format
- `http://www.opengl.org/resources/libraries/glut`

Section 11.2

OpenGL

Function Naming Conventions

- all OpenGL functions begin with `gl`
- some OpenGL commands have numerous variants that differ in number and type of parameters
- such commands are named using following pattern:

  $$generic\_name\ N\ T\ V$$

  where $generic\_name$ is generic name of function, N is digit (i.e., 2, 3, 4) indicating number of components, T is one or two letters indicating data type of components, V is either nothing or letter v to indicate component data specified as individual values or as vector (i.e., pointer to array), respectively

Number N

2	(x,y)
3	(x,y,z)
4	(x,y,z,w)

Data Type T

`b`	byte	`ub`	unsigned byte
`s`	short	`us`	unsigned short
`i`	int	`ui`	unsigned int
`f`	float	`d`	double

- `glVertex3f`: specific version of generic `glVertex` function that takes three `float` parameters
- `glVertex3fv`: specific version of generic `glVertex` function that takes single pointer to array containing three `float` values

Specifying Vertices

- vertices are building block for all geometric primitives (e.g., lines, triangles, quadrilaterals, polygons)
- vertex specified with `glVertex*` command
- although homogeneous coordinates employed, some functions allow specification of point using less than 4 coordinates (e.g., `glVertex2f`, `glVertex3f`)
- when two coordinates x and y given, refer to homogeneous coordinates $(x, y, 0, 1)$ (e.g., `glVertex2f`)
- when three coordinates x, y, and z given, refer to homogeneous coordinates $(x, y, z, 1)$ (e.g., `glVertex3f`)
- for example, each of following refer to point (1, 2, 0) :

```
glVertex2f(1.0, 2.0);
glVertex3f(1.0, 2.0, 0.0);
glVertex4f(2.0, 4.0, 0.0, 2.0);
```

Specifying Geometric Primitives

- all geometric primitives are specified by vertices
- use glVertex* to specify vertices
- use glBegin and glEnd to specify how vertices to be interpreted (e.g., individual points, pairs of vertices specifying line segments, triples of vertices specifying triangles)
- for example, code used to specify single point:

```
glBegin(GL_POINTS);
glVertex3f(1.0, 2.0, 3.0);
glEnd();
```

- for example, code used to specify triangle:

```
glBegin(GL_TRIANGLES);
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(1.0, 0.0, 0.0);
glVertex3f(1.0, 1.0, 0.0);
glEnd();
```

Specifying Geometric Primitives (Continued)

Value	Meaning
GL_POINTS	individual points
GL_LINES	pair of vertices interpreted as line segments
GL_LINE_STRIP	series of connected line segments
GL_LINE_LOOP	series of connected line segments with segment added between last and first vertices
GL_TRIANGLES	triples of vertices interpreted as triangles
GL_TRIANGLE_STRIP	linked strip of triangles
GL_TRIANGLE_FAN	linked fan of triangles
GL_QUADS	quadruples of vertices interpreted as quadrilaterals
GL_QUAD_STRIP	linked strip of quadrilaterals
GL_POLYGON	boundary of simple convex polygon

Additional Properties for Geometric Primitives

- each vertex has several properties in addition to position, including: color, normal vector, texture coordinates, material properties
- to specify color, use `glColor*`
- to specify normal vector, use `glNormal*`
- do not place calls to other functions other than ones specified above inside `glBegin` and `glEnd`
- for example, code used to specify triangle:

```
glBegin(GL_TRIANGLES);
glNormal3f(0.0, 0.0, 1.0); // set current normal (0,0,1)
glColor3f(1.0, 0.0, 0.0); // set current color to red
glVertex3f(0.0, 0.0, 0.0);
glColor3f(0.0, 1.0, 0.0); // set current color to green
glVertex3f(1.0, 0.0, 0.0);
glColor3f(0.0, 0.0, 1.0); // set current color to blue
glVertex3f(1.0, 1.0, 0.0);
glEnd();
```

State Management

- `glEnable` and `glDisable` used to enable and disable certain features/functionality

State Management

Value	Meaning
`GL_CULL_FACE`	if enabled, cull polygons based on their winding in window coordinates (e.g., do not render backs of faces)
`GL_DEPTH_TEST`	if enabled, do depth comparisons and update depth buffer
`GL_LIGHT_`i	include light i in evaluation of lighting equation
`GL_LIGHTING`	if enabled, use current lighting parameters to compute vertex color
`GL_LINE_SMOOTH`	if enabled, draw lines with antialiasing
`GL_NORMALIZE`	if enabled, normal vectors specified with `glNormal` scaled to unit length after transformation
`GL_POINT_SMOOTH`	if enabled, draw points with antialiasing
`GL_RESCALE_NORMAL`	if enabled, normal vectors specified with `glNormal` (assumed to be of unit length) scaled to unit length after transformation (only works for certain types of transformations)

Point Size and Line Width

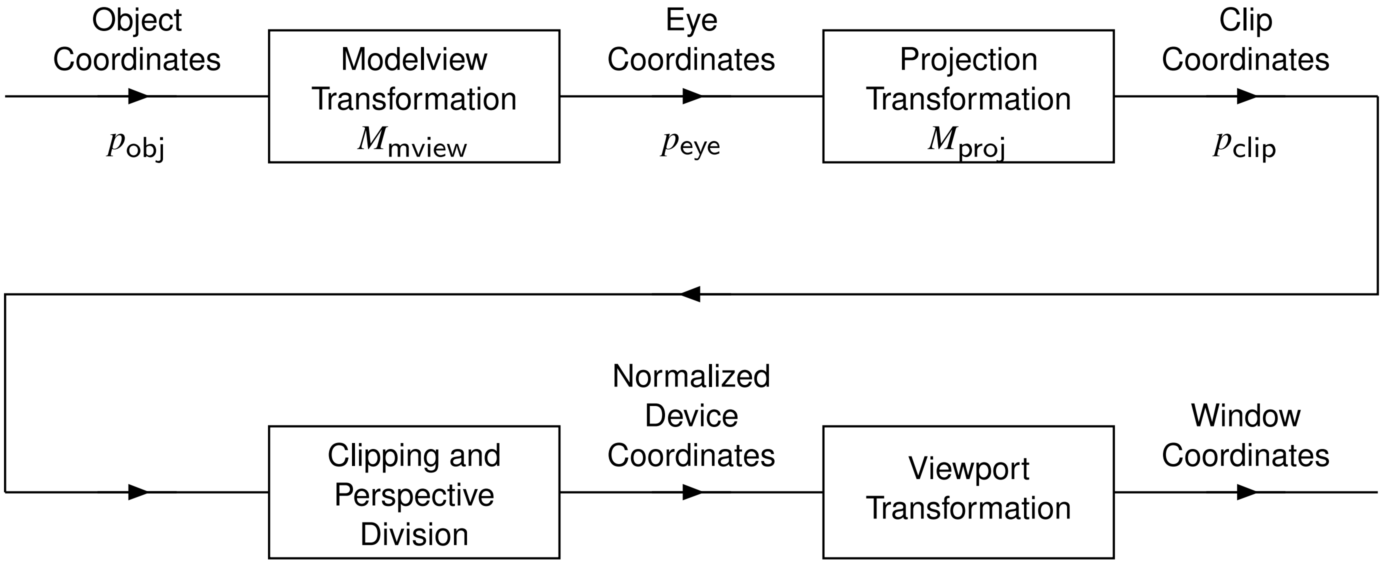
- for point, can specify point size (in pixels) with `glPointSize`
- for line segment, can specify line width (in pixels) with `glLineWidth`
- for example, code like:

```
glPointSize(10);
glBegin(GL_POINTS);
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(1.0, 0.0, 0.0);
glEnd();
```

- for example, code like:

```
glLineWidth(10);
glBegin(GL_LINES);
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(1.0, 0.0, 0.0);
glEnd();
```

Transformations


Object Coordinates
p_{obj}
Modelview Transformation M_{mview}
Eye Coordinates
p_{eye}
Projection Transformation M_{proj}
Clip Coordinates
p_{clip}
Clipping and Perspective Division
Normalized Device Coordinates
Viewport Transformation
Window Coordinates

- vertices specified in object coordinates
- $p_{\text{eye}} = M_{\text{mview}} p_{\text{obj}}$
- $p_{\text{clip}} = M_{\text{proj}} p_{\text{eye}}$
- eye positioned at origin looking in direction of negative z axis (with up direction in direction of positive y axis)
- use `gluLookAt` to simulate different eye position and orientation

Modelview and Projection Transformations

Function	Description
`glMatrixMode`	set current matrix
`glLoadIdentity`	load current matrix with identity matrix
`gluPerspective`	multiply current matrix by perspective projection matrix
`gluOrtho`	multiply current matrix by orthographic projection matrix
`gluOrtho2D`	multiply current matrix by 2-D orthographic projection matrix
`gluLookAt`	multiply current matrix by viewing transformation matrix
`glRotatef`	multiply current matrix by rotation matrix
`glScalef`	multiply current matrix by scaling matrix
`glTranslatef`	multiply current matrix by translation matrix
`glPushMatrix`	push current matrix on current matrix stack
`glPopMatrix`	pop current matrix from current matrix stack

Composition of Transformations

- functions that multiply current matrix always ***postmultiply*** (i.e., multiply on right)
- let M denote modelview matrix
- let p_{obj} and p_{eye} denote point in object and eye coordinates, respectively
- modelview matrix updating example:
  1. `glMatrixMode(GL_MODELVIEW);`
     `glLoadIdentity();`
     $M = I$
  2. `glRotatef(45.0, 0.0, 0.0, 1.0);`
     $M = IR_{\mathsf{z}}(45) = R_{\mathsf{z}}(45)$
  3. `glTranslatef(1.0, 2.0, 3.0);`
     $M = R_{\mathsf{z}}(45)T(1,2,3)$
- so, we have $p_{\mathsf{eye}} = R_{\mathsf{z}}(45)T(1,2,3)p_{\mathsf{obj}}$
- first, translate by $(1,2,3)$; then rotate 45 degrees about z axis

Viewport

Function	Description
`glViewport`	set viewport (i.e., drawable part of window)

Other Functions

Function	Description
`glClear`	clear buffer to preset values
`glClearColor`	specify clear values for color buffers
`glShadeModel`	select flat or smooth shading
`glFrontFace`	define front-facing polygons (e.g., projection is CCW or CW loop)
`glLight*`	set light source parameters (e.g., position, color)
`glLightModel*`	set lighting model parameters
`glColorMaterial`	specify how material color should track current color

Additional OpenGL References

- full documentation on each OpenGL function can be found at:

  `http://www.opengl.org/sdk/docs/man`

OpenGL Example Programs

- ***Simple 2-D Graphics* (`simple_2d.cpp`).** Draws a point, line, triangle, and quadrilateral.
- ***Simple 3-D Graphics* (`simple_3d.cpp`).** Draws and animates several simple polyhedra.
- ***3-D Graphics with Lighting* (`cube.cpp`).** Draws a cube with lighting.

OpenGL/CGAL Example Programs

1. ***Wireframe Mesh Viewer* (`wireframe.cpp`).** Allows a polygon mesh to viewed as a wireframe.