ELEC 486/586:
Multiresolution Signal and Geometry Processing with C++

Michael Adams

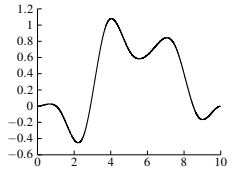
Department of Electrical and Computer Engineering
University of Victoria
Victoria, BC, Canada
E-mail: mdadams@ece.uvic.ca

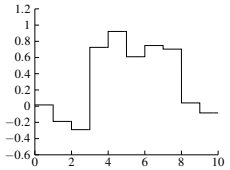
* *While waiting for the lecture to begin, please complete the initial course questionnaire.* *

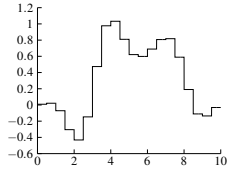
Course Overview

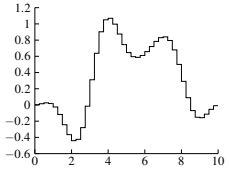
- objective: to give students experience in implementing software solutions to engineering problems in a manner consistent with what is done in industry in order to ***help students more easily find jobs*** upon graduation
- interdisciplinary in nature (e.g., engineering and computer science)
- consists of ***two highly-integrated half courses***:
  1. multiresolution signal and geometry processing
     - multiresolution processing deals with representations that capture information at different levels of detail
  2. C++ programming for engineers
     - C++ programming language, libraries, software tools, and related topics
- very strong emphasis on problem solving using C++
- cover various methods from signal and geometry processing and then ***implement*** methods ***using C++***
- ***no prior knowledge of C++ required*** (starts from beginning)

Approximations Using Haar Wavelet System


Original

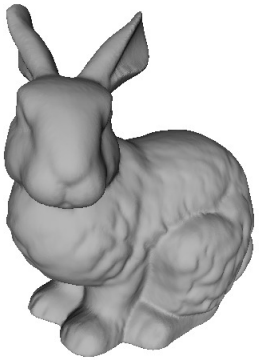

Coarse Approximation

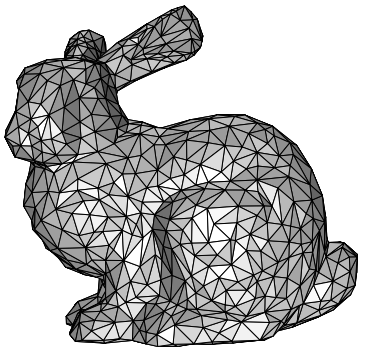

Medium Approximation


Fine Approximation

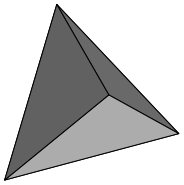
Geometry Processing

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

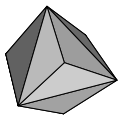

Surface


Polygon Mesh

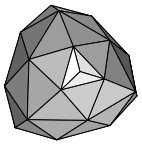
Subdivision Surfaces



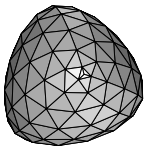
Control Mesh



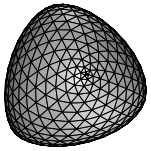
Refined Mesh
After One Iteration



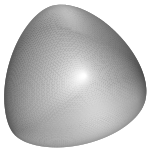
Refined Mesh
After Two Iterations



Refined Mesh
After Three Iterations



Refined Mesh
After Four Iterations



Limit Surface

Multiresolution Signal Processing

- multiple sampling rates employed
- study fundamentals of multirate signal processing and various multirate structures (e.g., sampling-rate converters, filter banks, transmultiplexers)
- sampling-rate converters:
  - audio/image/video processing
- filter banks:
  - signal coding/compression
  - denoising, restoration, enhancement, adaptive filtering
  - data encryption, error control coding
- transmultiplexers:
  - multichannel communication systems, multicarrier modulation systems (e.g., ADSL)
  - frequency-division multiple access (FDMA) 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)

Multiresolution Geometry Processing

- multiresolution: representations that capture different levels of detail
- study representation of surfaces using polygon meshes and subdivision methods
- applications:
  - computer graphics, animation
  - biomedical computing
  - computer-aided design and manufacturing
  - scientific visualization
  - finite element analysis, computational fluid dynamics

C++ Programming

- software development tools
  - compiler/linker (e.g., GCC C++ compiler), build tools (e.g., CMake), version control systems (e.g., Git)
- C++ programming language
  - basics, classes, templates
- C++ standard library
  - containers, iterators, algorithms, I/O streams, time measurement
- Open Graphics Library (OpenGL), a de-facto standard library for high-performance 3-D computer graphics
  - fixed graphics pipeline only (i.e., no shaders)
- OpenGL Utility Toolkit (GLUT), a popular cross-platform auxiliary library for OpenGL
- Computational Geometry Algorithms Library (CGAL), a library widely used in industry for geometric computation
- emphasis on industry-standard libraries

Course Design Considerations

- design constraints:
  - not just programming course
  - study signal/geometry processing methods
  - actually implement some methods in C++ before end of course
- problem:
  - must cover programming material in sufficient time to be able to solve non-toy problems (in signal/geometry processing) before end of course
  - must have all programming assignments completed before classes end for term without having multiple assignments crammed together at end
- solution:
  - equivalent of approximately 3 to 4 weeks of lecture material delivered in video-lecture form in order to front-end load instructional material
  - by allowing more programming material to be covered earlier, can avoid having multiple programming assignments crammed together at end
  - to compensate for material covered in video lectures, approximately last 3 to 4 weeks of lecture time slots converted from regular lectures to bonus lectures, covering variety of extra-credit topics
  - students not responsible for material covered in bonus lectures and attending bonus lectures is optional

Video Lectures

- core programming content *initially delivered* by video lectures
- then, regular (i.e., in-class) lectures focus on *more difficult aspects* of material from video lectures as well as answer student questions
- video lectures available via course instructor's YouTube channel:
  `https://www.youtube.com/user/iamcanadian1867`
- URLs for videos can be found on video-lectures handout
- must watch video lectures according to specified schedule
- must submit feedback questionnaire on each video watched prior to specified deadline
- *strongly recommended that students work through programming exercises as corresponding topics covered in video lectures*
- students *encouraged to work ahead* in video-lecture schedule in order to reduce workload later in term

Computer-Based Tutorial

- tutorial is not tutorial in usual sense employed by most courses
- scheduled in computer lab for access to C++ software development environment
- students work on *programming exercises*
- students can also work on *programming assignments*
- students have opportunity to ask for help with programming-related aspects of course
- tutorials start in *first* week of classes
- tutorials do not run for full duration of class schedule (only approximately 8 or 9 tutorials)
- *tutorial attendance is mandatory*

Course Outline Handout

DISCUSS THE COURSE OUTLINE HANDOUT.

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)
- easier to migrate from C++ to C, Java, and many other languages than other way around