C++

And DrawSVG, Scotty3D, cmake, IDEs...

Why C++?

- Graphics Ecosystem is almost entirely based on C++
  - Both industry & academia
- Can express high level constructs
- Can maintain high performance
- Interfacing with graphics APIs
- Why not Rust, JS, …?

Goals Today

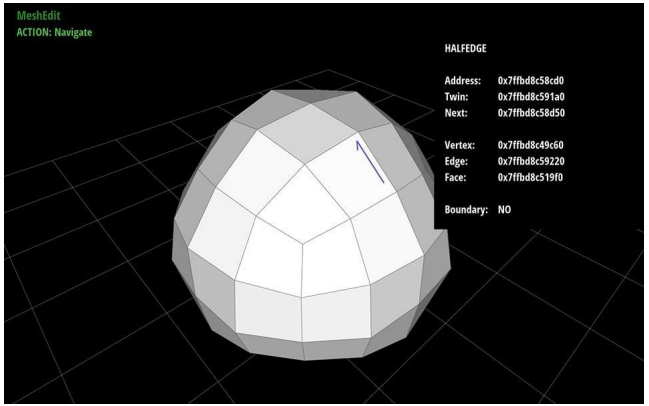
- Set up a C++ environment for DrawSVG/Scotty3D
- C++ crash course (coming from C)

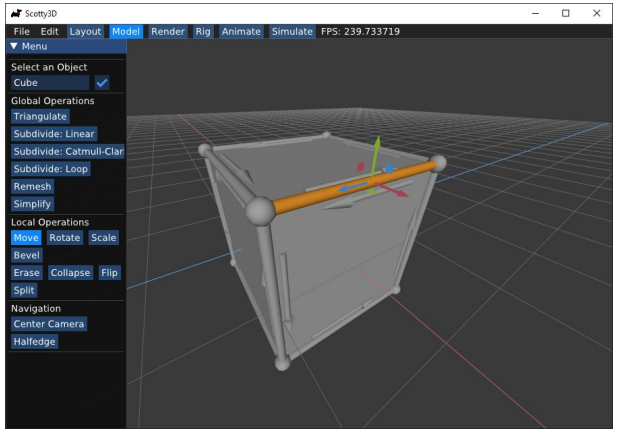
Answering Questions

Building DrawSVG/Scotty3D

Getting the Code

- Fork/clone/download the code skeleton
  - https://github.com/CMU-Graphics/DrawSVG (Assignment 1)
    - M1 = look at the build piazza post
  - https://github.com/CMU-Graphics/Scotty3D (Assignments 2-4)
- Scotty3D documentation is being updated
  - We will be pushing edits to documentation throughout the semester, so don't worry if A3 and A4 don't make sense yet!





Version Control

- Please make private repositories!
  - If you choose to push your repositories to Github or similar websites for storage, please make sure they are private repositories. We don’t want YOU or anyone else to get in trouble by violating academic integrity policies.
  - Please also make a repository! Local repositories are not always secure, so keeping a cloud version for backup is always a good idea.

Building the Code - Prerequisites

CMake and a C++ compiler: g++, clang++, or cl (visual studio)

- Linux/macOS
  - You should already have g++/clang++
  - Install cmake
    - If you are using apt, the version might be too old. If so, install cmake using pip
  - Install Scotty3D dependencies, e.g.
    - `brew install pkg-config sdl2`
    - `apt install pkg-config libgtk-3-dev libsdl2-dev`
- Windows
  - Download and install CMake
    - Remember to select “add to path”
  - Download and install Visual Studio Build Tools 2019
  - If you want the whole Visual Studio IDE, install Visual Studio Community 2019 instead

Visit https://cmu-graphics.github.io/Scotty3D/build/ for a more detailed explanation of all this.

Building the Code - CLI

- Linux/macOS

```
mkdir build
cd build
cmake ..
make -j5
```

Run with

```
./drawsvg ../svg/<path>/image.svg
./Scotty3D
```

Run `./Scotty3D -h` to discover command line arguments.

- Windows

```
build_win.bat
```

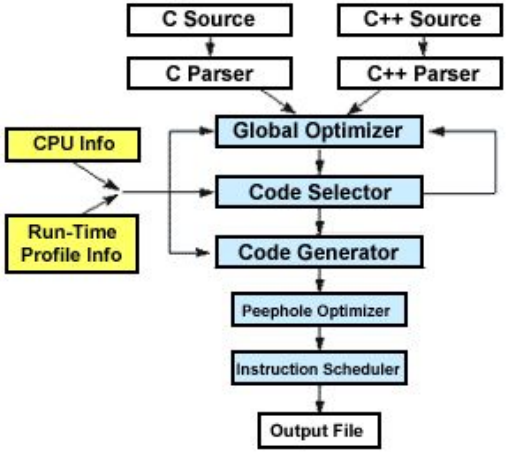
Or, with `cl` in scope:

```
mkdir build
cd build
cmake ..
cmake --build .
```

We've tested the build process on Windows, macOS, and a few Linux distros, but your setup might still break (that's C++). If you're having trouble, come to OH.

Debug vs. Release Builds

- Important!
- Debug builds disable most compiler optimizations
  - This makes your program easier to debug, but potentially unusably slow in some cases.
- Release builds are the opposite
  - Some debuggability can be maintained by using RelWithDebInfo mode
- https://cmu-graphics.github.io/Scotty3D/build/



Take 15-411/611

Tip: if you have (e.g.) a really old integrated laptop GPU and are getting < 30 FPS, try turning multisampling to 1 under Edit > Scotty3D Settings > Multisampling

Testing

- Important!
- We will be grading your projects on our own computers (more below)
- Make sure autolab can build your code
  - Some C math library behavior (used in pathtracer) may differ across compilers - use e.g. std::abs() instead of abs() to mitigate this. Use uint32_t, etc. instead of unsigned int.
  - Undefined behavior may play out differently
  - Don’t ignore warnings! What may be a warning for you may be a compile error for a TA because of the OS.

Warning: the visual studio compiler is unusually lenient regarding non-standard extensions and lacks some warnings (that we turn into errors) present in gcc/clang.

Setting up an IDE

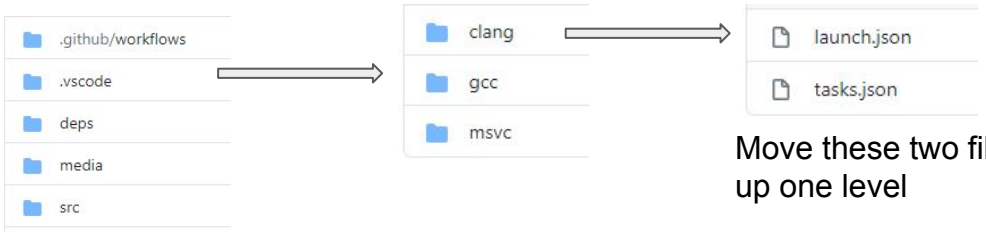
Why use an IDE?

- Autocomplete
  - Automatically find functions and variables within scopes without manually trawling through headers
- Graphical debugging
  - Much easier to navigate and understand than e.g. gdb command line
- Code navigation
  - Makes traversing a large codebase much easier with go to definition/implementation, symbol search, go to error, find references, etc.

You can set up these features in text editors like vim/sublime and operating systems like emacs as well, but we won't go over that today.

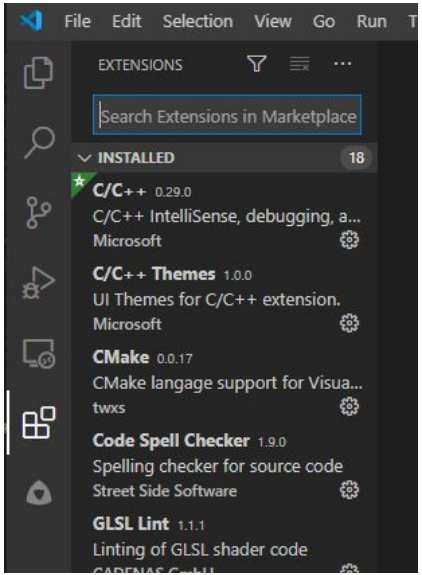
VSCode - all platforms

- This semester we're recommending VSCode on all platforms
- https://code.visualstudio.com/download
- You will need to install the "C/C++" extension
- Scotty3D comes with build/launch data in the .vscode folder
  - Move launch.json and tasks.json for your compiler into .vscode/
  - Open the Scotty3D folder in VSCode, and you should be able to build and debug with Ctrl+Shift+B and F5

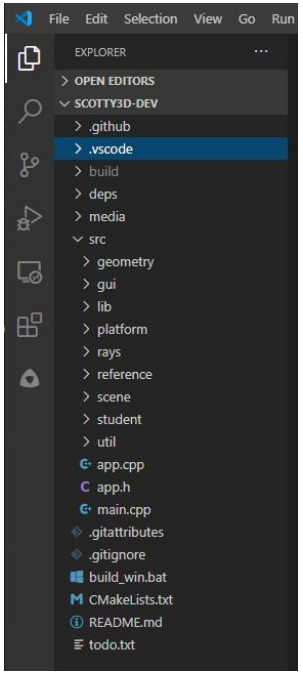


VSCode is technically another text editor, but the C/C++ extension gives us the IDE features we care about.

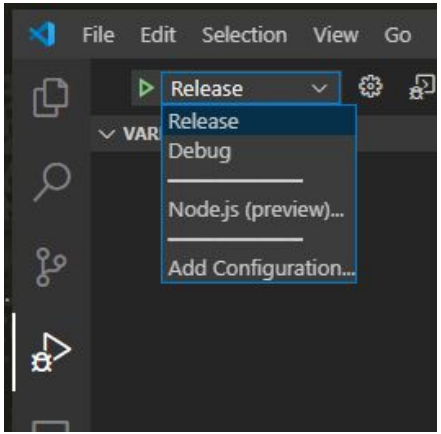
VSCode - UI



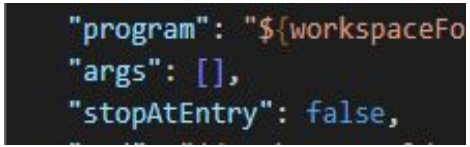
Install extensions



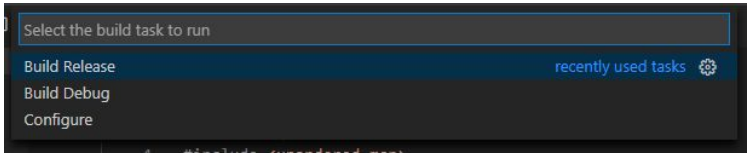
Browse files



Launch debugger



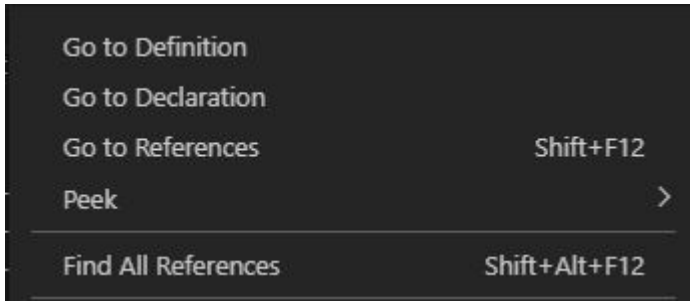
Launch.json: add command line arguments



Build project

VSCode - Basic Hotkeys

- Ctrl+Shift+B: Run build task
- F5: debug
  - F6: Pause, F10: step over, F11: step into, Ctrl+Shift+F5: restart
- Ctrl+Shift+P: run command
- Ctrl+P: go to file
- Alt+o: swap to header/implementation file
- Right click ->



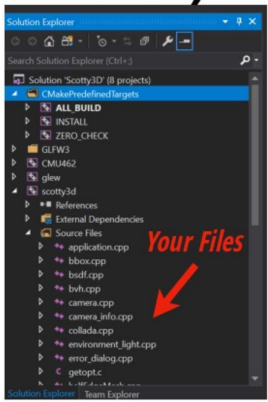
Setting up Visual Studio 2019 (Windows)

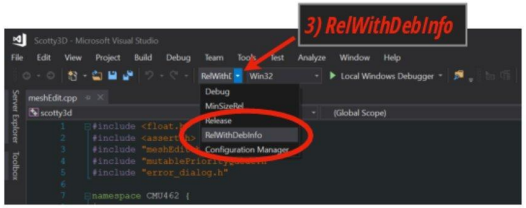
- Install the latest version of CMake on Windows: https://cmake.org/download/
  - Make sure you select "Add CMake to the System PATH for all users" during installation.
  - If you forgot to do add CMake to the PATH, see https://www.architectryan.com/2018/03/17/add-to-the-path-on-windows-10/
- Double click the file `runcmake_win.bat` at the root directory of Scotty3D
  - You should only need to do this once.
  - This will create a Visual Studio *solution* file for you in the `build` folder
  - (By the way, this just runs `cmake -G "Visual Studio 16 2019" ..`)

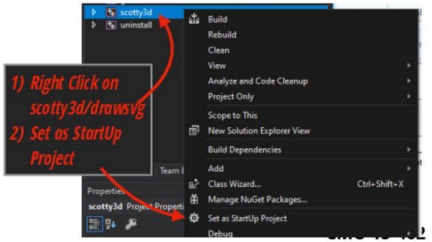
CMU 15-

Setting up Visual Studio 2019 (Windows)

- Double click on `DrawSVG.sln` inside the `build` folder (or `Scotty3D.sln` for A2,3,4)
  - Visual Studio will open with the project
- Navigate the files you need to edit via the Solution Explorer to the right
- Right click on "drawsvg" (or "scotty3d" for A2,3,4) and click "Set as StartUp"
- Change the "Solution Configuration" from "Debug" to "RelWithDebInfo" or else Scotty3D will run without optimizations (slow as molasses)



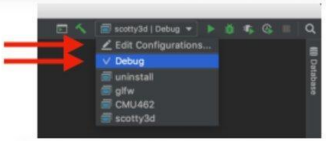


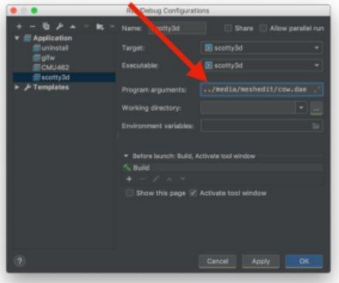


Slide from two years ago - details at
http://15462.courses.cs.cmu.edu/spring2020/lecture/intro-cpp/slide_013

Windows/Mac/Linux: CLion

- Cross Platform IDE for C/C++ Development.
  - Free for students with a @andrew.cmu.edu email: https://www.jetbrains.com/student/
- Simply open the DrawSVG root directory, it should detect the CMake project out of the box
- To add command line arguments click on the configuration dropdown on the top right, then Click Debug. Then open it again and click "Edit Configurations..."





Slide from two years ago - details at
http://15462.courses.cs.cmu.edu/spring2020/lecture/intro-cpp/slide_008

For Mac Users: XCode

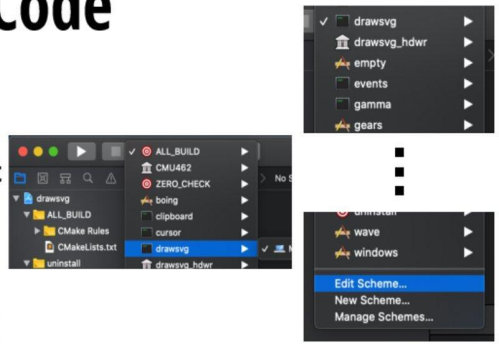
- Xcode is Apple's IDE for the Mac. It supports C++, Objective-C, and Swift development, and has some editors specific to building Mac and iOS apps.
  - Download free from the Mac App Store (search "Xcode")
- CMake can create Xcode projects for you out of the box.
  - From the root directory of Scotty3D:

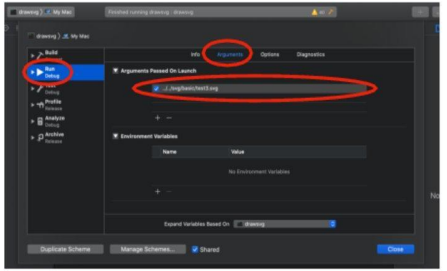
```
$ mkdir build
$ cd build
$ cmake -G Xcode ..
```

- Now double click on `drawsvg.xcodeproj` to open in Xcode

For Mac Users: XCode

- To Set Command Line arguments:
  - Click on the "Scheme" ALL_BUILD at the top of the screen, change to drawsvg, then click "Edit Scheme"
  - Navigate to "Run" on the left, then "Arguments", then add the svg you want to open (or folder of svgs), like "../../svg/basic/test3.svg" or "../../svg/basic/"
  - Click "Close"
- To build / run, click on the Arrow on the top Left of the window.

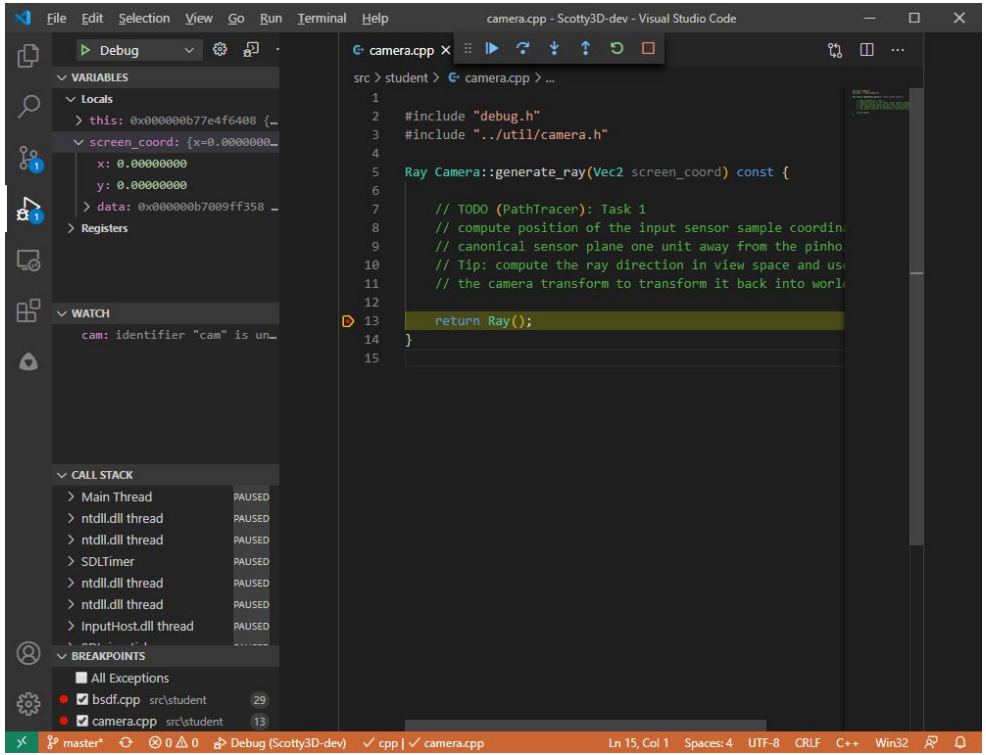




Slide from two years ago - details at
http://15462.courses.cs.cmu.edu/spring2020/lecture/intro-cpp/slide_009

Debugging

- Debugging doesn’t have to be as painful as using the GDB command line
- All of the environments talked about have some key features
  - Breakpoints + Conditional breakpoints
  - Stop on exception
  - Watch variable values / single step / pause / play
- Profiling
- Tutorials are easy to google



C++

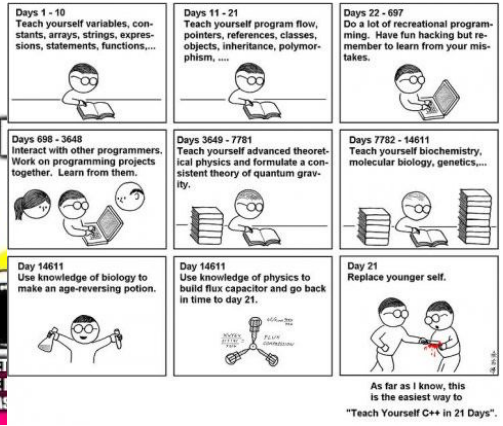
C++

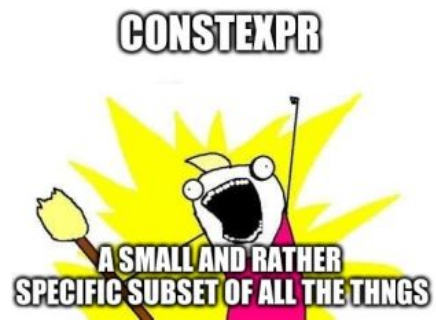
C++11

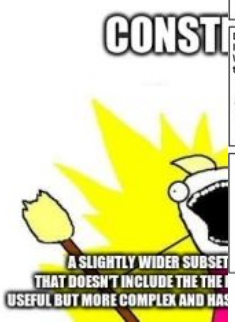
C++14

C++17

C++20





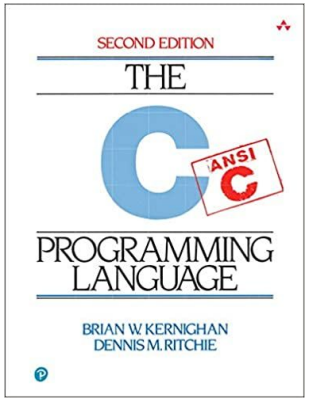


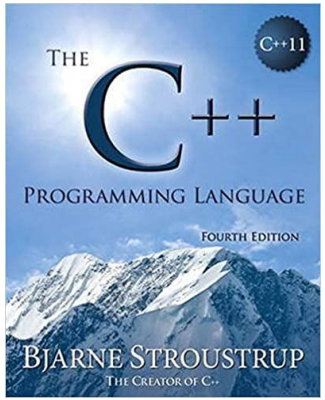
In C++ we don't say "Missing asterisk" we say "error C2664: 'void std::vector<block,std::allocator<_Ty>>::push_back(const block &)': cannot convert argument 1 from 'std::_Vector_iterator<std::_Vector_val<std::_Simple_types<block>>>' to 'block &&'" and i think that's beautiful

4:30 PM - 1 Jun 2018

Beautifully short error message ^

From C to Shining C++





- C++ is (nearly) a superset of C
  - You can write C-style code
  - But not recommended

- Useful C->C++ transition links
  - https://berthub.eu/articles/posts/c++-1/
  - https://isocpp.org/faq
    - ‘C++ if you already know C’, 'General Topics', 'Classes and Objects', 'Container Classes'
  - https://en.cppreference.com/w/
    - General reference
  - This presentation

Basics

- Default parameters
- Overloading
- auto

https://ideone.com/XTGhk3

```
void f( int x, double m=12.34 );
int main(){
f( 1, 1234.56 ); // ok
f( 1 ); // ok, m=12.34
f(); // error! no default value for x
}
```

```
25 auto EnergyDifferential(auto directions,auto i,auto j,auto Edir,auto E)
26 {
```

Returning to C++ after months of programming in Python

Namespaces

- Allows us to define things (functions, types) with the same names
- Useful for libraries and/or segmenting portions of the codebase
- Access a namespace with the '::' operator
- Bring an entire namespace into scope with 'using'
- Example: standard library is gated behind std::
- Example: Scotty3D path-tracer code is in PT::

https://ideone.com/nNnpll

Strings

- std::string encapsulates c-style strings
- Still null-terminated
- Dynamically allocated (with some exceptions)
- Supports operations like copying, concatenation, substring searching, etc.

https://ideone.com/QitXUw

Streams

- Just saw the canonical way to print in C++
- This generalizes to “streams”
- cout, cin
- Files, strings
- Can define how the stream operators ‘<<’ and ‘>>’ work for custom types - later.

https://ideone.com/Wx91vE

References

- Restricted version of pointers - can’t be null (and shouldn’t be invalid, but…)
- Must be initialized to refer to a real object
  - Can’t be re-bound to refer to anything else
- Allows “pass by reference”
  - Like passing a pointer, does not copy the parameter
  - Harder to pass in invalid data than with a pointer
  - Later: const reference
- Just like appending ‘*’ to make a pointer type, you append ‘&’ to make a reference type.

https://ideone.com/8kC2vM

Classes

- The objects in object oriented programming
- Collections of data + functions to operate on the data
  - Important: constructors & destructors
  - Copy constructors, later: move constructors
- Can hide data/functions such that only other class members can access them
  - Actual encapsulation
  - In C++, structs & classes are the same except for default visibility
- Also basis for inheritance & dynamic polymorphism, but we won't go over that
- Examples: halfedge mesh, scene object, pathtracer
  - You won't need to create new classes, but it may be useful

https://ideone.com/J367Eb

Initializers: wtf? https://www.youtube.com/watch?v=7DTlWPgX6zs

Allocations

- Don’t use malloc & free
- May use new & delete
  - Runs constructors/destructors on allocated storage
  - Types can also implement custom new/delete behavior
- Try to use data structures that manage their own storage instead
  - Later: STL data structures
  - Also: unique_ptr, shared_ptr, optional

https://ideone.com/WXR7EQ

Const

- Const is a lot more important in C++ than C
- const values
  - Same as C, value is immutable
- const references
  - Also similar to const pointers in C, but with reference semantics
  - Canonical way to pass no-copy immutable parameters
- const member functions
  - Able to be called on const objects

https://ideone.com/sBL0CA

Operator Overloading

- Define your own operators
  - Math +,-,*,/,&,|
  - Comparison ==,!=,<,>,<=,>=
  - Deref/call/index ->,(),[]
  - Streams <<,>>
  - Assignment =
- Can make code less clear
  - Most useful for math-like operations on custom types
  - Scotty3D: vector & matrix math
- Special name syntax, otherwise exactly the same as normal functions
  - The = operator is extra special

https://ideone.com/iYqFyc

Templates

- Allows generic programming
- Run-time vs. compile-time
- Type parameters
- You can go super deep with template metaprogramming and compile-time evaluation (constexpr)
    - All you really need to know here is how to instantiate templated data structures. Later: STL data structures
    - Will implement some templated member functions in Scotty3D

https://ideone.com/ZQv4pW

Standard Library Data Structures - Arrays

- Theme: Resizable, iterators may be invalidated when the data changes
- **std::vector<T>**
  - Variable-length array holding Ts
- **std::unordered_map<K,V>**
  - Hash map from Ks to Vs
  - Needs to know how to hash Ks (automatic for primitive types)
  - Not ordered
- std::unordered_set<T>
  - Unordered map but only keys
- std::stack<T>, std::queue<T>, std::priority_queue<T>

https://ideone.com/DegrkC
https://en.cppreference.com/w/

Standard Library Data Structures - Trees

- Theme: Iterators don’t get invalidated when the structure changes
- std::map<K, V>
  - Ordered dictionary (usually a red-black tree)
- std::set<T>
  - Map but with only keys (still a red-black tree)
  - Can be used as a priority queue that supports erase in log(n) time
- std::list<T>
  - Doubly linked list
  - Used in MeshEdit halfedge mesh

https://ideone.com/ty8ekQ
https://en.cppreference.com/w/

Iterators

- Generalization of pointers specific to iterating data
- Allows for traversing data, how depends on the type
  - Forward/backward
  - Random access
- Starts at data_structure.begin(), ends at data_structure.end()
  - data_structure.end() is **past** the last element, and is returned from searching functions to signify “not found”
- Can be invalidated! What if the element it refers to is deleted or moved?
- Range-based for loops
- Will have to use iterators in MeshEdit
  - HalfedgeRef, VertexRef, etc. are all iterators

https://ideone.com/ZsXDHm

Standard Library Algorithms

- Get iterators from data structures and feed them to algorithms
- std::sort(start, end)
  - Generally useful
- std::lower_bound(start, end, value)
  - Also exists as a member of ordered containers
  - Useful for path tracer
- std::find(start, end, value)
- std::partition(start, end, comparison_function)
  - Useful for path tracer
- std::swap(l, r)

https://ideone.com/g4vrE0

Move Semantics

- Unless you use a reference, passing around an object will copy it many times
- This might not be desired behavior
  - What if you want to “give away” an object so that some other piece of code owns it?
- Lvalues vs Rvalues
- Rvalue references
- Overload functions to take Rvalue references = create copy vs move functions
- Shouldn’t have to do moves in your code (just use references), but moves are used liberally in the skeleton code

https://ideone.com/zR4PMl

Lambda Functions

- std::function<R(Ts…)> represents a function type
  - Can be a variety of underlying representations
- You can make higher-order functions
- Lambda syntax defines a function value inline
- Captures
  - Value, reference, this
- Parameters
- What is the actual type of a lambda?
  - Need to use auto or std::function

https://ideone.com/mKeLRk