CS 354R: Computer Game Technology

http://www.cs.utexas.edu/~theshark/courses/cs354r/
Fall 2017
Instructor and TAs

• Instructor: Sarah Abraham
  • theshark@cs.utexas.edu
  • GDC 5.420
  • Office Hours: MW4:00-6:00pm or by appointment
• TA: TBD
Communications

• We’re using Piazza for class communication
  • Announcements, issues, and questions, etc
  • Students should work together before asking for teacher or TA involvement
  • Enroll at http://piazza.com/utexas/fall2017/cs354r

• Grades and assignments will be done via Canvas, http://canvas.utexas.edu/
Books and Resources

- Recommended “textbook”: “Game Engine Architecture” Jason Gregory
  - Good exposition of many engine technology and design
  - Not required but useful
- Other useful books:
  - “Game Programming Gems 1-8”
  - “3D Game Engine Design” David Eberly (lots of equations, less exposition, good math background and computer graphics)
- Website: www.gamasutra.com
  - Game developer technical and trade news
- GDC Vault and Siggraph archives
Class Expectations

• During class we’ll explore key concepts and provide basic background info for projects
  • Ideally a time for discussion and group exploration
  • Participation and questions highly encouraged!
• Outside of class you’ll implement this functionality in your game engine
  • Note that this is a programming-heavy course!
• Throughout the course you will encounter new technology and ideas that I won’t teach directly
Things I Will Skim Over

• 3D graphics concepts and programming
• Vectors, matrices, geometric reasoning
  • OpenGL will be the graphics API discussed in lectures, but we won’t work at that level
• C++ programming
• UI toolkits (FLTK, Glut, Qt, Interface Builder, etc)
• Scripting languages (Lua, Python, etc)
Grading

• Projects and reports (no tests)
  • 1 – 2 major projects, broken into phases and graded separately
  • Small groups of 3 assigned by the TA
  • Self-forming groups allowed for the final project
• Potentially periodic quizzes to check comprehension and attendance
Working in Groups

- Working in groups is an acquired skill
- Most important thing you’ll learn in here
- For some information on group functioning, read http://www-honors.ucdavis.edu/vohs/index.html
- We assign teams — like in industry
- Group evaluation exercises throughout the semester will ensure an even distribution of work (and grades)
  - Continued low performance can result in failing the class
Grading

- Groups will be graded as one, but adjustments will be made based on individual performance
- Each group will set milestone goals for the current phase
  - One to two milestones per phase
- You will be graded based on how well you achieve your goals factoring in degree of difficulty
- Each milestone will involve turning in an artifact/demo
Project Tools

- Projects must run and be turned in on the 64-bit Linux machines in the GDC basement
- We use Ogre3d as the 3d engine
- We’ll recommend sound packages, UI packages, etc. but you can choose your own within limits
- We use Bullet for physics
- You can develop on your own machines, but code and demos are tested on the lab machines
- You will spend a surprising amount of time getting your tools installed, working, and playing together. This is part of the problem solving aspect of this course
- Source code control systems are essential for team projects
Tools for Content Creation

- Models and art are the biggest expense in real games
- This course doesn’t require outside art assets, but:
  - You can use Blender in the lab or on your own machines
  - Acknowledge any assets you download/purchase
  - Assets must be usable in the Linux environment but you can develop in non-Linux environments
- Be prepared to write small tools if you think it will make your project easier
- Be prepared to write format converters if you have a good tool that produces output that your game engine can’t input. This is a big deal in the real world as well as in class!
What Is Game Technology?

- Technology that drives games
  - Graphics
  - Physics
  - GUI
  - Networking
  - AI
  - Sound
- Game engine connects these aspects in a coherent, organized manner
What This Course Is Not

• Not a game design course!
• Not a game development class!
• Thus making a game with cool systems is secondary to creating the engine that drives them
  • But game features and systems are of course closely connected to engine implementation…
The Computer Game Industry

- Hardware makers produce gaming hardware
  - Sony, Nintendo, Microsoft, NVIDIA, etc
- Engine teams develop underlying libraries to build upon
  - Epic, Unity, GameMaker, etc
- Game developers create games using these engines
  - Insomniac, Level 5, Arkane, etc
- Publishers release games to players
  - Sony, Nintendo, EA, Steam, etc

Note that a company can do one or more of these (e.g. Blizzard)
Game Development Team

- System designers decide on game format and behavior
- Artists create models, textures, and animations
- Level designers create the game spaces and interactions
- Audio designers handle sounds
- Programmers write code to put everything together and create tools to make everyone else’s job easier
- And others: production, management, marketing, quality assurance
Interactive Programming

- A game is a user-controlled program
- Responsive to user input in real time
- Help users understand what is happening at all times
- Provides constant, up-to-date feedback about its state (and user input)
- Effective interaction is key for immersion

- How should we structure our software to achieve this?
Event-driven Programming

- Everything happens in response to events
- Events can occur asynchronously with respect to the execution of the program reacting to the event
- Events can come from users or system components
- Generated signals or messages sent to a system component
  - So in some sense events, signals, messages are equivalent
System-generated Events

- Timer events
  - Application calls a function requesting an event at a future time (e.g. next time a frame should be drawn)
  - System provides an event at the requested time
  - Application checks for and responds to the event (e.g. drawing the next frame)
User-generated Events

- User presses a button on a joystick
  - Joystick hardware sends a signal to the computer (called an interrupt)
  - The OS responds to the interrupt by converting it to an item in an “event queue” for the windowing system
  - Events can be kept in priority order, temporal order, etc
  - API elements of UI toolkits check and respond to events

- How should our windowing system check for events?
Polling vs. Waiting

• Can provide a call that returns immediately (nonblocking) to check if an event is pending
  • Happens whether or not there is an event
• What do you do if there’s not one?
  • Loop to keep checking? Go off and do something else?
• Also possible to use a blocking event function that waits (blocks) until an event has arrived
  • Only returns after the event is processed
• What happens while your program waits?
  • Does any work get done? Does the screen freeze up?
Callbacks

• Tell system what to do when a particular type of event arrives
  • Necessary code now executes automatically
• Most GUI systems operate this way
• Application makes a call to the GUI to tell it what function should be executed when the event arrives
  • When a timer event arrives, the system calls a draw function
  • When the left mouse button is clicked, the system calls the mouse event function
Event-response Classes

- Two fundamental kinds of event responses:
  - Mode change events
    - Cause the system to shift to a different mode of operation
  - Task events
    - Cause the system to perform a specific task within a mode of operation
- Game software structure reflects this
  - Menu system is separate from game runtime, for instance
Real-time Event Loops

- Games and similar interactive systems look like an big infinite loop:

```c
while (1) {
    process events
    update state
    render
}
```

- The number of times this loop executes per second is the frame rate (since each render operation creates a new frame)
- Measured in frames per second (fps)
Latency and Lag

• Latency is the time it takes from starting to do something to finishing it
• Lag in user interaction is the latency from when a user provides input to the time they see the response
• Controlling lag is extremely important for playability
  • Distorts causality
  • Causes motion sickness
  • Makes it hard to track or target objects
  • Makes interaction difficult
Computing lag

Frame time
- process input
- update state
- render

Frame time
- process input
- update state
- render

Frame time
- process input
- update state
- render

\[ \text{Event arrives} \]

\[ \text{Lag} \]

\[ \text{Time} \]

\[ \text{lag}_{\text{max}} \approx \frac{2}{fr} \]

\[ \text{lag}_{\text{avg}} \approx \frac{1.5}{fr} \]
How Can We Reduce Lag?
Brute Force

1. Pick a frame rate = 1/frame time
2. Do as much as you can in a frame time
   • Faster algorithms and hardware means more can get done!
   • Budgeted resources – graphics, AI, sound, physics, networking, etc — must now be done in the frame time

• Is this necessary for all resources?
Prioritizing Resources

• Most important to reduce lag between user input and its direct consequences
  • Lag between input and other consequences may matter less
• Update different parts of the game at different rates
  • Achieve this by decoupling separable parts of the game