GPAT – CHAPTER 5 AND 6 INPUT AND SOUND

INPUT

INPUT DEVICES

- **Digital** input is binary (on or off)
  - Button on controller
  - Key on keyboard
- **Analog** input has a range of values
  - Joystick
  - Trigger
- Games often need to deal with
  - **Chords** – multiple simultaneous inputs
  - **Sequences** – series of inputs

DIGITAL INPUT

- Essentially simple Boolean checks

```
if isPressed(INPUT) then
    changeGameState()
```

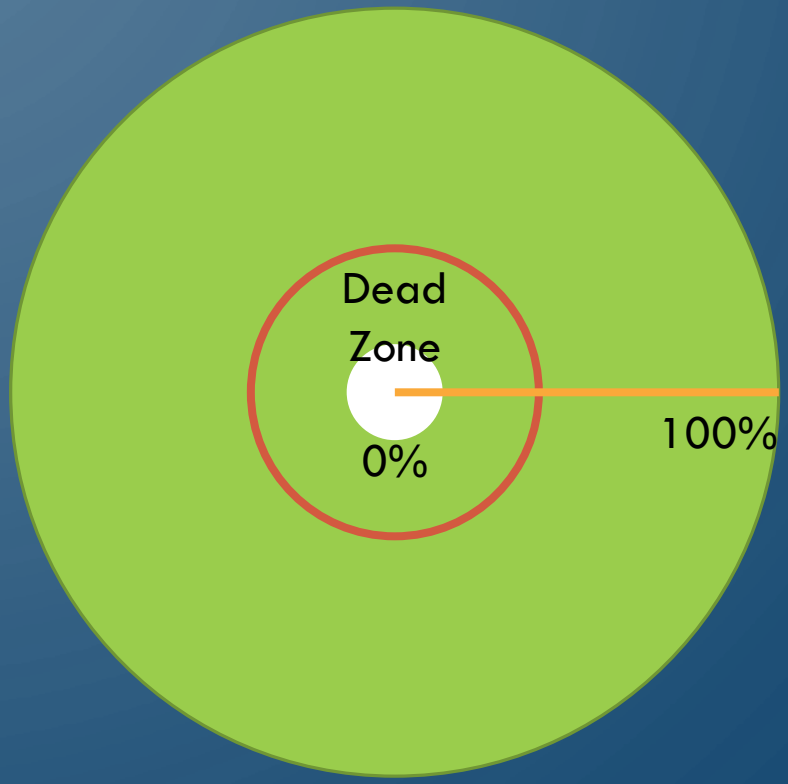
- INPUT is often referred to as a **keycode** (from keyboard lingo)
- Problem is that this might register over multiple frames
- How can you deal with and program this?
  - Separate push and release actions and respond on state changes
  - Switch! Why not if/else?

ANALOG INPUT

- Often represented as a multi-bit integer, e.g., 16 bit (-32768 – 32767)
- Need to deal with error in input device
  - Example – a joystick at rest might not have a value of 0, but a value close to 0

ANALOG INPUT

- To deal with error implement **analog filtering**
  - Example – implementing **dead zone** (do nothing) around center of joystick
- Simple range check with conditional
  - Be sure to use length and not raw x and y values. Why?
  - After dead zone, renormalize range of valid input. Why?



EVENT SYSTEMS

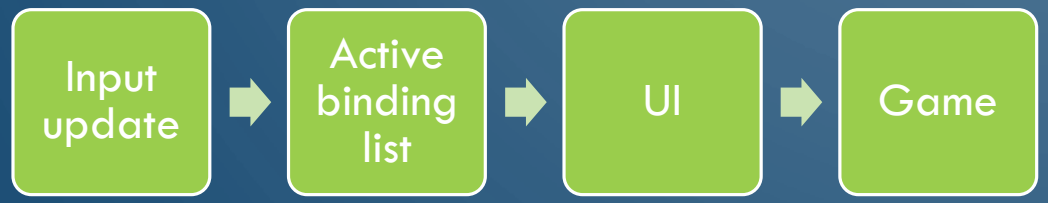
EVENT SYSTEMS

- Prior, we looked at a **polling** system where we checked each frame the state of input
- **Event systems** essentially are a **push** notification system. In these, we "register" to an event notification
  - Registration literally links a method (often called a **callback**, **handler**, or **slot**) to an event (often called a **signal**)
  - The underlying event system must still implement itself through polling! However, it is encapsulated in the event handler (good OOP design!)

A BASIC EVENT SYSTEM

- For starters, if you were never aware:
  - functions (and methods) are literally stored in memory (where?)
    - Also can use function objects
  - We can have variables refer to them called function/method pointers (examples?)
- In event manage class store list of methods registered to an event
  - Provide a method to register handlers
  - Update will:
    - Poll
    - On event, invoke each method registered
- Example of mouse click event
  - ```
    // Accept function with
    // specific signature
    register(function handler(int, int))
      callbacks.add(handler)
    ```
  - ```
    processInput()
      if mouseClicked then
        for each Callback c ∈ callbacks do
          c(mouseX, mouseY)
    ```
  - Implement manager class with **singleton** pattern
    - A class designed for and accessed through a single instance

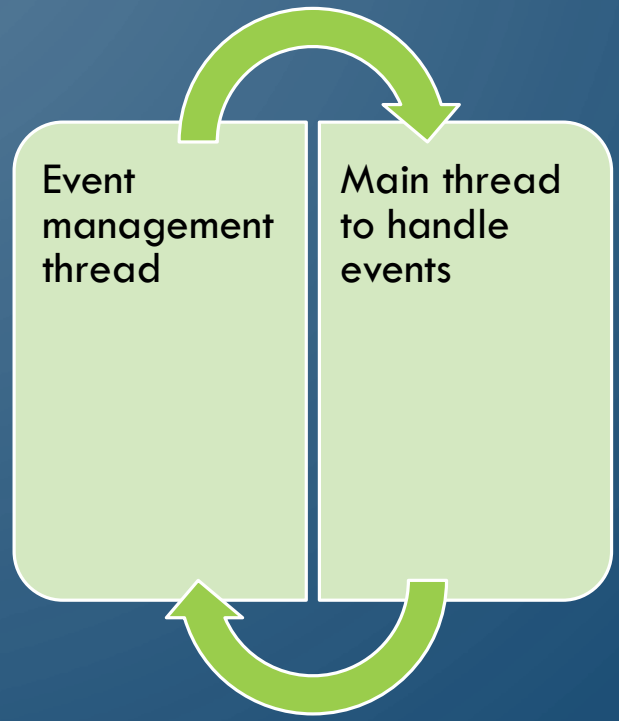
A MORE COMPLEX EVENT SYSTEM



- Generalize from a specific button or keypress to an abstract action
- Actions are methods bound to an event (called a **binding**, i.e., a registered action)
- To process input, poll the system to gather the active bindings. After, send list of active bindings to UI first and then to the game state
  - How should we store the bindings and active bindings?
  - Why do we send to the UI first?

A PLACE FOR MULTI-THREADING

- Often event-based systems are multi-threaded
  - One thread for the event management
    - All polled events go onto an event queue
  - One thread for handling the events
    - Process all events currently on the queue (or all within a limited time)
- This has an advantage that input can be captured live, i.e., when a player performs the input
- Often at the OS level at least

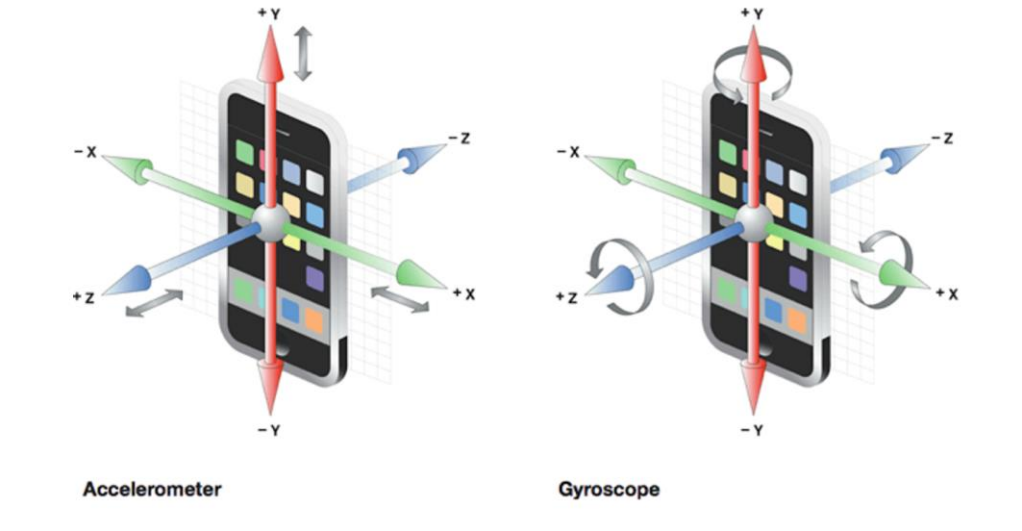


MOBILE INPUT

TOUCH SCREENS AND GESTURES

- Player interacts with finger (similar to mouse clicking)
  - Complicated by **multi-touch** (multiple finger input) and **gestures** (series of touch actions)
- Many gestures are readily available through libraries, however you can often design your own
  - Analyze gestures using the **Rubine algorithm** which analyzes and matches **features** of a gesture

OTHER INPUT

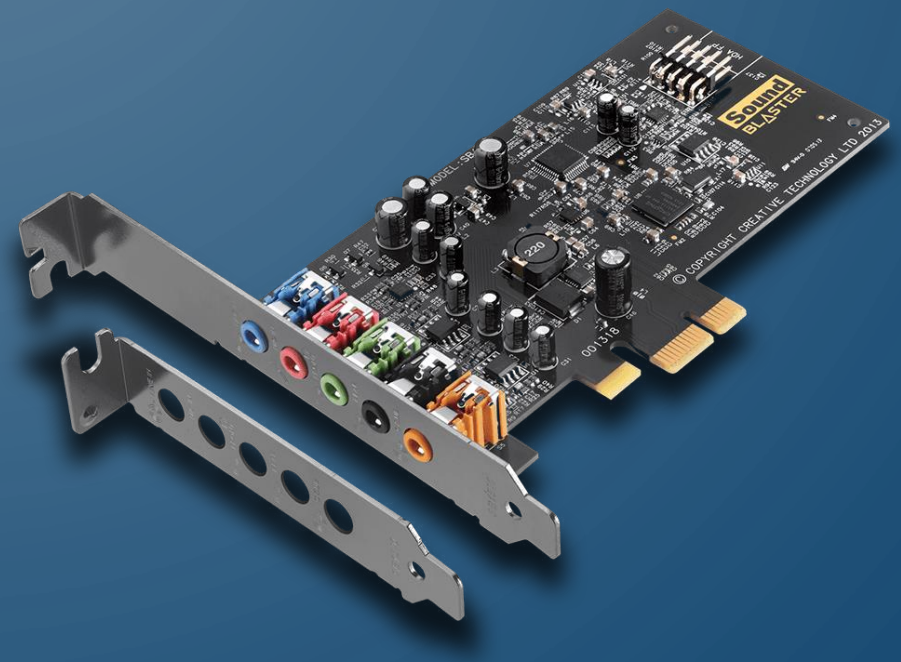


- An **accelerometer** detects acceleration along the coordinate space represented by the device with itself at the origin
- A **gyroscope** measures rotation around the devices principle axes

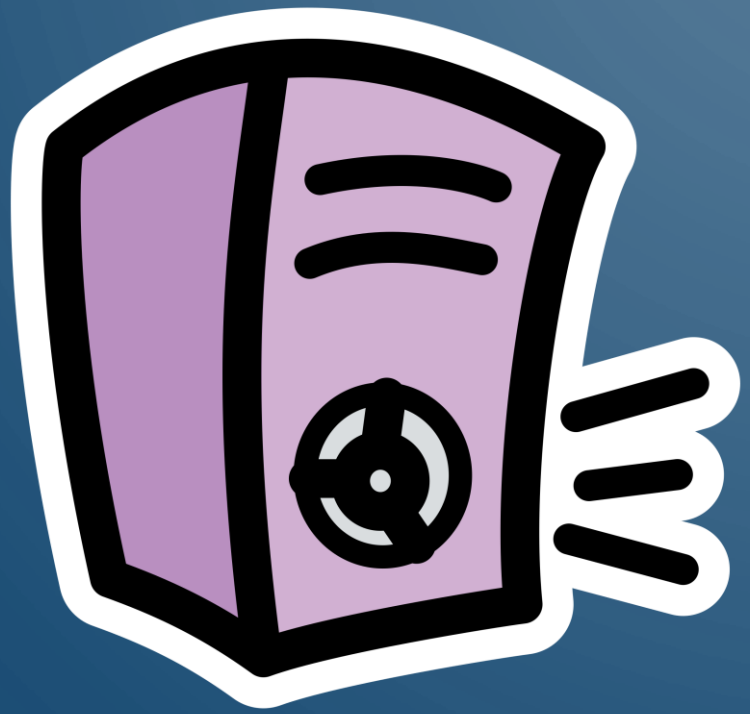
SOUND

BASIC SOUND

- Games need to playback standalone sound files
  - There is a limited number of **channels** or simultaneous sounds that can be played at a time
- Source data
  - Audio files
  - Stored in or streamed from local memory to the sound card
    - Data transferred from CPU memory to sound card memory through memory **buffers**



BASIC SOUND

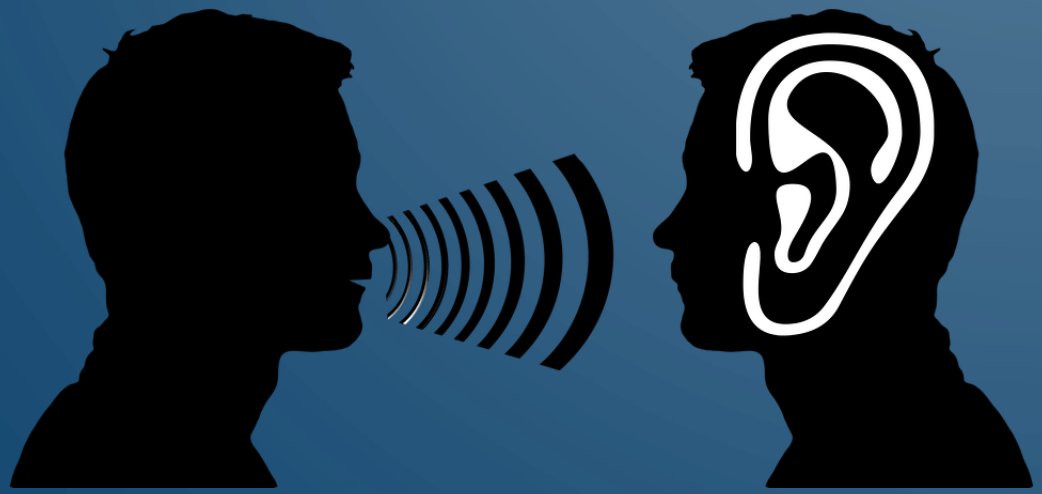


- Sound cues indicate an action or trigger for a sound, called a sound event
  - Often multiple sounds are associated with each cue to provide variety
    - Randomized
    - Location-based
  - Meta-data about the sound
  - Manager would be vary similar to an event-based system

2D VS 3D SOUND

- 2D sound is typically positionless sounds that play equally out of left and right speakers
  - Example: background music or UI sounds
- 3D sound takes into account position and orientation of a **listener** and multiple sound **emitters**
  - Volume of sound is determined based on distance between them (**falloff**)

LISTENER



- Need to be careful on position and orientation of listener
  - Could choose camera position and orientation, e.g., first-person views
  - Might be better to choose a position other than a player avatar, e.g., in third-person

SURROUND SOUND

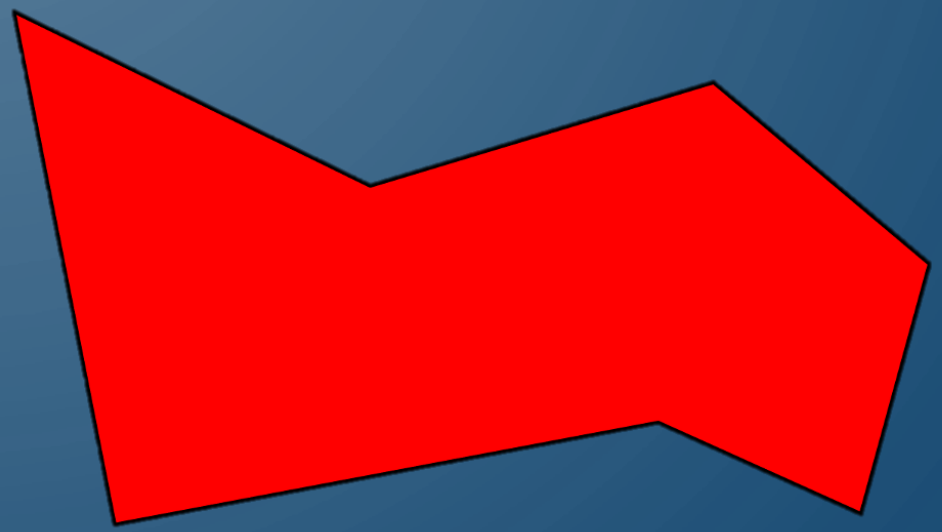
- With 3D sound you have an additional difficulty of deciding volume of sounds presented to each speaker (left vs right)
- What about surround sound, should you design for it? Pros/cons?

SOUND PROCESSING

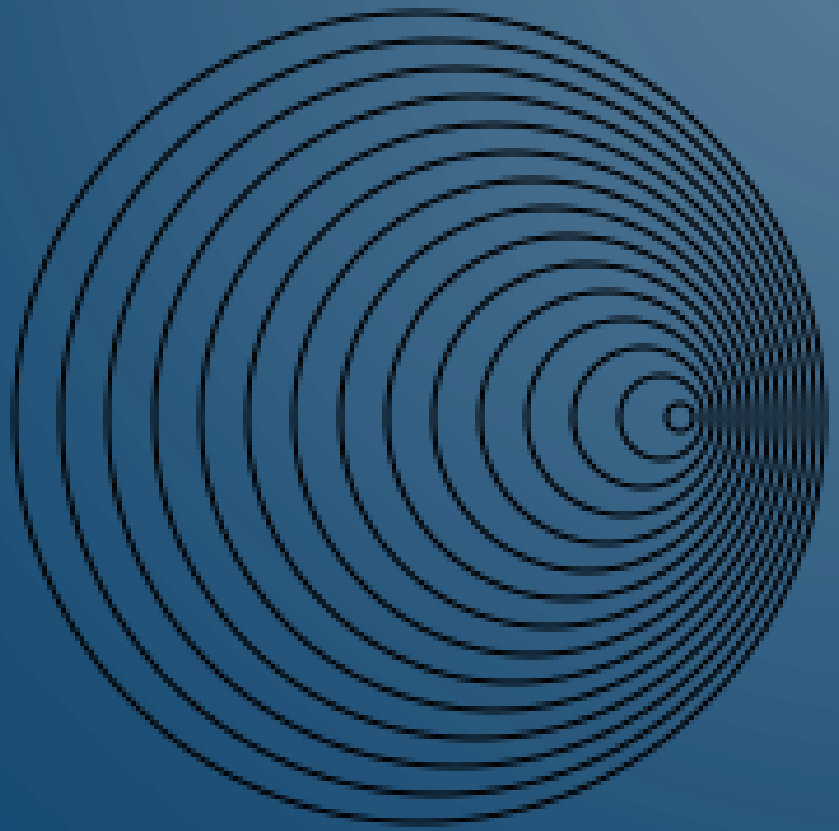
- **Digital signal processing** is the computational manipulation of sound
  - Example: **reverb** or echoing
  - Example: **pitch shift** – alters sound frequency
  - Example: **compression** – volume modification to normalize sounds
  - Example: **low-pass filter** – reduces volume of high pitch sounds

SOUND PROCESSING

- Can provide local modifications based on location in virtual world
  - Will discuss more geometry in Ch. 7 with physics



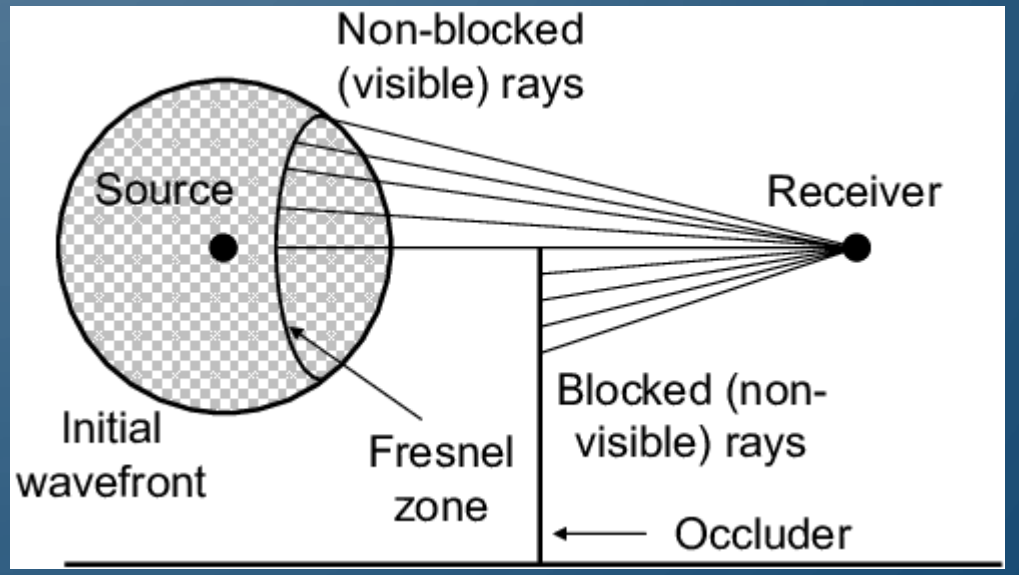
DOPPLER EFFECT



- Pitch increases on approaching sounds
- Pitch decreases on receding sounds
- Caused by variation on time taken for sound waves to travel
- Can also be applied to lighting in games (e.g., outerspace settings)

OCCLUSION AND OBSTRUCTION

- One difficulty with sound is that it reflects off of obstacles and refracts through them
  - Computationally intensive to mimic physics
- Two considerations that can often be managed
  - **Occlusion** occurs when there is not a direct path from listener to emitter
  - **Obstructions** occurs when sound might not have a straight-line path
  - Can use **Fresnel Acoustic Diffraction** to compute



SUMMARY

- Event systems manage matching input actions to callback functions
- Sound is complex and involves many design decisions
  - Libraries make programming with it much simpler