Minecraft3 Feedback

• Looks great!
• For those of you that didn’t finish, that’s okay because...
We’re Pivoting

- Brown Minecraft Club
- Minecraft4 instead of Platformer1
- Need some time to write support code / figure out how much we’re changing Platformer
LECTURE 6
User Interfaces
User Interfaces

USER INTERFACES
Many shades of UI

• Heads-up display (HUD)
  – Persistent 2D elements drawn over the game
    • Ex. your health & resources, minimap, available actions/cooldowns

• Orthographic UI
  – 2D elements drawn in the 3D scene
    • Ex. entity health bars, tips, popup dialog, 2D special effects
    • Usually doesn’t scale with camera distance

• Separate screens/overlays
  – Inventory, menus, shops, dialog
Heads-Up Display

- Heads-Up Display (HUD)
  - Game stats (KDA, fps)
  - Minimap
  - Player health & abilities (most important)
  - Player stats & inventory

League of Legends
HUD Clarity

• The HUD takes up minimal screen space
• Each different type of information is placed in a distinct and isolated area of the screen
HUD Clarity

• Deciding where to put information (and how to group it) is very important
  – In League, there can be many passive effects on your character at once

‘Other’ effects go here

‘Battle’ effects go here
HUD Clarity

• But League is a battle arena (MOBA) game, so almost every effect is battle-related
  – The developers had to split them somewhat arbitrarily, which caused a lot of confusion
Drawing HUD

• Use the built in m_camera->setUI(true) so that your draw calls act in screen space

• Draw things over your game!
Orthographic UI

Tooltips

Health bars & Damage

(World of Warcraft)
Orthographic UI

“MEDIC!”
(Team Fortress 2)

Dialog
(Runescape)
How to Orthographic UI

• Goal: draw orthographic UI next to / above an entity within your game

• Approach
  1. Figure out screen space position of the object you want to draw
  2. Figure out if the game object is in front of or behind the camera
  3. Draw orthographic UI at that location (same way to draw HUD)
Converting to screen space

```cpp
vec3 convertToScreenSpace(vec3 pos, vec2 screenSize) {
    // get these from your camera
    mat4x4 view, projection;
    // this is necessary for the matrix multiplication
    vec4 fourVec = vec4(pos.x, pos.y, pos.z, 1.f);
    // project the position into clip space
    fourVec = projection * view * fourVec;
    // x and y range from -1 to 1
    vec3 clipSpace = vec3(fourVec.x / fourVec.w, fourVec.y / fourVec.w, fourVec.z);
    // convert x and y to pixel coordinates, leave z alone
    return vec3((clipSpace.x + 1) * .5f * screenSize.x,
                 (1 - clipSpace.y) * .5f * screenSize.y, clipSpace.z);
}
```
Using the projected position

• Use the x and y components of the vector calculated in the previous slide to position draw calls on the screen

• If the z-component is less than 0, it means that the object is behind the camera
  – In other words, don’t draw it!

• If it’s greater than 0, draw the UI!
Users Interfaces

ENGINE INTEGRATION
How to integrate UI?

Recommended:

• GameWorld has a UISystem with UIComponents.
• Tell your UISystem to draw after drawing all the other components
• If an orthographic UI component requires some game logic, have it belong to (and have a reference to) a GameObject.
Your UI

• You are required to implement some use of orthographic UI and HUDs in Minecraft 4
  — Try to think of something unique and interesting to your game concept!
  • E.g. inventory, health bars
More UI

• UI that is related to your GameWorld
  – In game pop-ups, shopkeeper menus, etc.
• UI that’s not related to your game
  – Settings menu, pause menu, etc.
• How do you manage this UI
  – Within your GameWorld?
  – Outside of your GameWorld?
  – Up to you
More UI

• Common elements
  – Buttons
  – Text elements
  – Containers

• For this week, required to have some basic UI outside of your game
3D-Integrated UI

Instructions
(Splinter Cell)

Minimap
(Far Cry 2)
3D-Integrated UI

All user interfaces as in-game holograms

(Dead Space)
Sound

BASICS
Sound in Games

• In the real world, computers have sound
• Background music
• Sound effects
• Can be an important part of gameplay
  – Listening for footsteps
  – Dramatic music
Sound File Formats

- Many ways to encode and store sound
- Open standards
  - Ogg Vorbis
  - FLAC
- Closed standards
  - mp3
  - m4a
  - wav
Sampled Audio

- mp3, wav, and most other familiar extensions
- Usually recordings of live sounds
- Samples of sound wave at regular intervals
- Prevalent in modern games
- Refers to data type, not origin
  - Touchtone telephone is generated but still sampled
Generated Audio

- **MIDI**
- File provides information on instruments and notes
  - Similar to sheet music
- Sound cards translate from instruments/notes to sound
- Can instruct computer to play something even if you can’t play it
- Used to be popular to save space, not as common now
Compressed vs. Uncompressed

Compressed Sound Files
- Lossy or Lossless?
  - Lossy remove “least important” parts of sound wave
  - Lossless just use smart compression on raw wave
- Smaller file size (esp. lossy)
- Lossy is lower quality
- Slower to decode and play
- Often used for music

Uncompressed Sound Files
- Record as much as possible of sound wave
- Much larger file size
- Usually high quality
- Faster to decode and play
- Often used for sound effects
Buffering

- Decompressing and decoding is slow
- Read sound into buffer, play back from buffer
- Size of buffer depends on speed of system
- Playback delay while buffer is filled
C++ LIBRARIES

Sound
Qt Sounds

• The Qt framework provides a few classes for basic sound playback:
  – QSound (general, best for music)
  – QSoundEffect (low latency sound effects)
• Easy set up – no need for external libraries!
• Read more here:
FMOD: What is it?

• Fully featured sound engine with support for 3D games
• Works on most platforms: Windows, OS X, Linux, Xbox, Playstation, and more
• Integrated as the primary sound system in Unity, Unreal Engine 4, and Source
• Used by Guitar Hero, BioShock, World of Warcraft, Dwarf Fortress, and many others!
FMOD: Features

• Supports three-dimensional sound playback and attenuation based on listener position and look vector
  – Just add a few extra lines to your Camera class

• In addition to sound effects and streaming, it supports sound generation, granular synthesis, recording, and more!

• See some examples here: http://www.fmod.org/documentation/#content/generated/common/lowlvel_examples.html
FMOD: Difficulties

- Need to set up an external library, which can be difficult if you haven’t done it before
- Supporting multiple platforms can be tricky, e.g. if you want to develop for both Windows and Linux
- The C++ API only works with the Visual Studio compiler on Windows, so you’ll have to use the C API instead (which is more annoying to use)
- Confusing to get into: lots of different technologies that you don’t really need (Studio, Designer, Event Player)
  - All you need is the Low Level Programmer’s API
FMOD: Recommendations

• Create your own sound interface that wraps all FMOD calls
  — This way, group members don’t have to know any FMOD, they just have to know your API

• Spend a good chunk of time researching how to set it up
Which library to use?

Qt Sounds
• No external library setup needed
• Simple, intuitive API
• Only supports “2D” sound playback
• No extras

FMOD
• Harder to setup, especially for multiple platforms
• More complicated API
• Supports 3D positional sound with attenuation
• Tons of extras, supports everything sound related
FMOD implementation

FMOD_SYSTEM *m_system;
FMOD_SOUND * m_music;
FMOD_CHANNEL * m_channel;
float m_soundLength;
float m_currChannelTime;
unsigned int m_length;
float m_pitch;

//initialize system
FMOD_System_Create(&m_system);
FMOD_System_Init(m_system, 100, FMOD_INIT_NORMAL, NULL);

// load the music file
FMOD_System_CreateSound(m_system, (std::string("meme.mp3").c_str(), FMOD_DEFAULT, 0, &m_music);

// get length in seconds if you need
FMOD_Sound_GetLength(m_music, &m_length, FMOD_TIMEUNIT_MS);
m_soundLength = ((float)m_length)/1000.f;

//play
FMOD_System_PlaySound(m_system, m_music, NULL, false, &(m_channel));
FMOD_Channel_SetFrequency(m_channel, 44100.0f * m_pitch);
FMOD_BOOL state;
FMOD_Channel_GetPaused(m_channel, &state);
if (state == 1) // music is paused
    FMOD_Channel_SetPaused(m_channel, 0); // start playing

// pause
FMOD_BOOL state;
FMOD_Channel_GetPaused(m_channel, &state);
if (state == 0) // music playing
    FMOD_Channel_SetPaused(m_channel, 1); //pause
Alternatively: OpenAL

- OpenGL style API for rendering 3D sound
- We don’t really recommend it – FMOD is better and more widely used
- From Wikipedia: “While the OpenAL charter says that there will be an "Architecture Review Board" (ARB) modeled on the OpenGL ARB, no such organization has ever been formed and the OpenAL specification is generally handled and discussed via email on its public mailing list.”
Types can be ugly

- Let’s say we have a class called BigBoy
- A smart pointer to a BigBoy would be `std::shared_ptr<BigBoy>`
- A vector of these would be `std::vector<std::shared_ptr<BigBoy>>`
- And an iterator pointing to somewhere in this list: `std::vector<std::shared_ptr<BigBoy>>::iterator`
Enter auto

• The keyword `auto` is magical: it makes the compiler *deduce* the type itself

• Super useful for avoiding long type names and improving legibility
int main(int argc, char **argv)
{
    std::shared_ptr<BigBoy> b = std::make_shared<BigBoy>();
    std::vector<std::shared_ptr<BigBoy>> vec;
    vec.push_back(b);

    // copy_of_b is of type "std::shared_ptr<BigBoy>"
    auto copy_of_b = vec.at(0);
    // b_deref is of type "BigBoy"
    auto b_deref = *copy_of_b;
    // bbIt is of type std::vector<std::shared_ptr<BigBoy>>::iterator
    for(auto bbIt = vec.begin(); bbIt != vec.end(); ++bbIt) {}
}

MINECRAFT3 PLAYTESTING!
To the Sunlab!