Lecture 10

Announcements
You’re done! (Almost)

• You’ve completed the main portion of the course!
  – Except retries, close enough
• Meeting after class in groups to discuss final ideas
• This week you will make a timeline and rubrics with your mentor TA
• From the final handout: engine requirements must at least be fully stubbed out
  – Interfaces and empty methods for engine requirements exist
  – Project must not crash
Special Topics Lectures

• We will try to cover the “big” features first
  • These are not intended to give you the full implementation details
    – Think “high-level overview”
    – You will have to do your own research
    – You’re focusing on these 1 or 2 things for 2 weeks, so make sure you’re interested in them

• Don’t wait for us to teach you – start your research now!
  – The staff has pretty good coverage, but even we don’t know some of this stuff
  – Teach us!
Introduction

QUESTIONS?
LECTURE 10

Sound
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 Sound Files

**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
Sound

C++ LIBRARIES
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](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
  – https://katyscode.wordpress.com/2012/10/05/cutting-your-teeth-on-fmod-part-1-build-environment-initialization-and-playing-sounds/
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
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.”
Lecture 10
Advanced Graphics
Advanced Graphics

PARTICLES
What is a particle?

- A particle is a tiny entity that is used in massive quantities to create a visually pleasing effect.
- Usually don't affect the gameplay in any significant way.
- Commonly used for explosions.
Particles

• What makes particles look good?
• Fade out or get smaller linearly over their lifespan
  – Once the particle is completely gone or transparent, it can be removed from the world
• Adding some kind of randomness to how they move
  – Starting position, velocity, acceleration, color, size, shape
Particles

• Particles are great
• But they are very slow if not done correctly
• Things that make them slow:
  – It’s a lot of information to tick
  – It’s a lot of information to draw
  – There are way too many of them to consider doing collision detection against each other
Particles

- Optimizations? Get ready.
- Reduce the amount of information in your particles
  - Vector3 position, Vector3 velocity
  - maybe some noise values to make them scatter
  - Less information to tick
- Don’t make your particles Physics Entities
  - Keep them in a separate list so that you can tick and draw them all at once
  - Binding the particle texture once and then drawing all your particles without unbinding and rebinding the texture is a HUGE improvement
- Don’t collide them with each other
  - That’s a lot of math
  - If you must have your particles collide, have them collide only with entities or terrain, not with each other
  - This means they also don’t need a shape, so they take up less space
- Keep them in an array
  - This limits the number of particles you can have (which is probably a good thing)
  - Keeps all the memory contiguous
  - Once you are trying to allocate more particles than you have room for, the oldest ones are kicked out first
  - Figure out how many particles your game can safely handle before a drop in framerate, and then limit your array size to about 80% of that number
- Tick them in your draw loop
  - But you said to never do that!
  - I know, sorry – but in this instance the speed gain of only having to iterate over them once is worth the terrible design
Particles

• What shape should my particles be?
  • Sphere?
    – Is limited if you want transparent particles
    – Too many vertices
  • Quad!
    – Use a texture like this one
    – Rotate the quad to always face the camera
    – This texture has a black background and no alpha information
      • Use glBlendFunc(GL_ONE, GL_ONE)
      • This says take all of the background, and add the color of this particle on top of it
      • Particles that are denser will appear brighter
QUESTIONS?

Particles
Motivation

• Per-vertex lighting is ugly
  – Can increase number of vertices, but this requires lots of extra memory
• Per-pixel lighting looks a lot better
  – But is much slower than per-vertex, must calculate lighting equation for each pixel instead of each vertex.
  – Naïve implementation has many wasted calculations
    • Calculates lighting on pixels that are later overwritten by a closer triangle
• Deferred lighting removes most of the wasted calculations and provides further optimizations
• Deferred lighting is an optimization, not a lighting model. You still have to choose a lighting model (for example Phong Lighting)
How can we avoid wasted calculations?

- Only calculate lighting once for each pixel
- But the fragment shader has no way of knowing if the value it is calculating will be the final value for that pixel
- Solution: Multiple passes

First pass
- Render geometry and keep track of data necessary to calculate lighting

Second pass
- Calculate diffuse and specular values that are independent of material

Third Pass
- Combine diffuse/specular from second pass with geometry and material (Object color for example) to complete lighting model
• A “pass” just means generating a texture (or multiple textures).
• Use framebuffer objects (FBOs) to group textures
  – FBOs are basically a collection of textures
  – The FBO allows you to write to these textures (instead of writing to the screen)
  – Default framebuffer is the screen
    • glBindFramebuffer(GL_FRAMEBUFFER,0)
  – See: glGenFramebuffers, glFramebufferTexture2D, glDrawBuffers
• For example:
  – First pass writes to “Texture1” (using “FBO1”)
  – Second pass reads from “Texture1” and writes to “Texture2” (using “FBO2”)
  – Third pass reads from “Texture2” and writes to the screen
First Pass

• Takes in all our geometry as input
  – Doesn’t need material properties (ex. textures)
  – This just means you need to draw everything in the scene using your first pass shader

• It outputs exactly the information we need to calculate lighting
  – Normals
  – Positions
  – Shininess – store as alpha channel of normal
Second Pass (1/2)

- Takes in normals, shininess and positions from first pass and light data
- Outputs diffuse and specular light contributions
  - Can save space by rendering to a single texture and storing specular contribution as alpha (but we only get monochromatic specular highlights)
  - Or render diffuse and specular to separate textures
- How do we send light data to GPU?
  - For each light:
    • Set necessary uniforms (position, direction, color, etc...)
    • Naïve: render full-screen quad to run the fragment shader on each pixel
      - Can do better, see slide 10
  - But each light would overwrite data from previous light
    • Solution: `glBlendFunc(GL_ONE, GL_ONE)` for additive blending
Third Pass (1/2)

- Takes in diffuse and specular contribution from second pass and geometry, textures, etc... (whatever we need to calculate object’s diffuse color)
- Render the scene again, this time applying any materials and finishing the lighting equation (i.e. finish calculating diffuse and specular term and add ambient + diffuse + specular)
- Output is our final lit scene which goes to the screen
Third Pass (2/2)
Optimizations

• Instead of calculating lighting on every pixel for every light, calculate lighting on only the subset of pixels that a light can possibly affect.

• Restrict the lighting calculations to the geometric shape that represents the volume of the light. In the second pass render this 3D shape instead of a full-screen quad.
  – Point light: sphere (radius usually based on attenuation)
  – Spot light: cone
  – Directional light: full-screen quad

• What if the camera is inside the shape of the light?
  – Represent light as a full-screen quad

• We will still have some wasted calculations, but this is much better (especially for small lights).
Optimizations

How the light is represented

Deferred Lighting – 11/18/2014
Optimizations

Diffuse contribution

Visualization of lights

Final scene

Deferred Lighting – 11/18/2014
Optimizations

- The second pass needs to know the position of each pixel in world space
  - Our first pass shader can easily write this position to a texture
- Doing this uses an extra texture (i.e. twice as much memory)
- Instead, can use the depth buffer from the first pass.
  - First pass already uses a depth buffer, so we don’t need any additional space.
- Depth buffer has z value from 0 to 1 for each pixel in screen space (convert x/y to screen space based on width/height).
- Use the (x,y,z) triplet in screen space and the inverse projection and view matrices to transform to world space.
Deferred Shading

• Deferred shading is another method for speeding up per-pixel lighting, almost the same as deferred lighting.
  – Note that the word “shading” here doesn’t refer to interpolating lighting values, it’s just the name of this technique
• Uses only 2 passes
  – First pass renders geometry storing normals as in deferred lighting, but also stores all material properties (diffuse color, for example) in one or more additional textures
  – Second pass calculates lighting and uses material properties to calculate final pixel color
• Pros: Less computation (don’t need to render the scene twice)
• Cons: Uses more memory and bandwidth (passing extra textures around)
Overview

• Deferred Lighting
  1. Render normals, positions, and shininess to textures
  2. Calculate diffuse and specular contributions for every light and output to textures
  3. Render scene again using diffuse/specular light data to calculate final pixel color (according to lighting model)

• Deferred Shading
  1. Render normals, positions, shininess, and material properties to textures
  2. Calculate lighting for every light and combine with material properties to output final pixel color
Disadvantages of Deferred Rendering

- Can’t easily handle transparency (this is a generic issue with z-buffer rendering techniques)

  Solutions:
  - Naïve: Sort transparent objects by distance
    - Slow
    - Can’t handle intersecting transparent objects
  - Order-independent transparency: Depth peeling
  - Use forward-rendering for transparent objects
    - Forward-rendering is the standard rendering pipeline that you’ve been using

- Can’t use traditional anti-aliasing techniques
  - MSAA (Multisample anti-aliasing), one of the most common AA techniques, doesn’t work at all with deferred lighting
  - Usually use some sort of screen-space anti-aliasing instead (FXAA, MLAA, SMAA)
Deferred Lighting

QUESTIONS?
Volumetric Effects

• Volumetric glow (fake scattering)
Volumetric Effects

- Volumetric glow (fake scattering)
  - Blend glow color over every pixel
  - Fade off using closest distance from light source to line segment starting from eye and ending at object under pixel
  - Requires deferred shading for position of object

- Rendering to entire screen is expensive
  - Fade off to zero at some radius
  - Only need to draw pixels within that radius in world space, will be cheap for a far away effect
  - Render using inside-out sphere with that radius
Advanced Graphics

SHADOW MAPPING
Shadow Mapping

- Need to test whether a pixel is in shadow
  - Render scene from light's point of view
  - Depth map stores closest point to light
  - Render the scene from the camera, projecting each point back into the light's view frustum
  - Shadow if depth of projected point > depth map
Shadow Mapping

• Need to fit frustum to light
  – Directional light =>
    parallel rays =>
    orthographic
  – Spot light => frustum =>
    perspective
  – Point light => rays in all
    directions => use cube map
Shadow Mapping

• Problem: Jagged edges
  – Shadow map resolution varies across scene
  – Increasing resolution helps, but uses more memory
Shadow Mapping

• Fix: blur or fuzz out boundaries
  – Multiple nearby shadow tests are made per pixel and are averaged together
  – Called PCF: Percentage Closer Filtering
    • May use randomized sample patterns
    • May use variable blur size since shadows get more blurry away from caster and area lights
Shadow Mapping

• Fix: Average out over multiple frames
  – Reproject previous frame (if moving camera)
  – Jitter shadow map per frame for more samples
  – Weight by confidence (distance to texel center)
Cascaded Shadow Maps

• Shadow mapping has problems
  – Resolution varies across scene
  – One shadow map per object doesn't scale
• Idea: fit several shadow maps to camera
  – Want uniform shadow map density in screen-space
  – Objects near eye require higher world-space density than objects far away
  – Use a cascade of 4 or 5 shadow maps
  – Each one is for a certain depth range of the scene
• Used in almost all modern games
Cascaded Shadow Maps

FIGURE 4.1.2 The view frustum in world space split into three cascade frustums and their corresponding shadow map coverage. We use a top view with the light direction pointing straight down the horizontal world plane.
Cascaded Shadow Maps

• Use depth in shader to choose cascade
  – Can blend between two closest cascades to smooth out discontinuities

Scene with a cascade of 3
Questions
Lecture 10
Tips for Final Design
Extra credits!

• The minimum viable product:
  – [https://www.youtube.com/watch?v=UvCri1tqlxQ](https://www.youtube.com/watch?v=UvCri1tqlxQ)

• This is what the first \(\sim\) 2-3 weeks of your final project should be

• You will have to add some polish during the later weeks, but the ideas here still apply
LECTURE 10

C++ Tip of the Week
C++ Tip of the Week

MUTABLE
Mutable

• Declares that a variable can be changed, even in the `const` scope of a function or object:

```cpp
string::length() const
{
    if( !isValidLength ) // boolean, initialized to false on object creation
    {
        dataLength = strlen( data ); // can't be changed if not mutable
        isValidLength = true; // same
    }
    return dataLength;
}
```
Mutable

• Why bother marking the method `const`?
• The user expects `length()` to have no side effects
• Generally, it’s okay to lie to the user like this as long as the function keeps the public-facing information constant
• This is very useful for objects that cache data
You should warn users if the function is expensive, since its runtime isn’t always the same.

`mutable` can also be used with `lambda` functions, because by default any variables captured by value cannot be changed.

Thanks to mjm9 for this week’s tip.
C++ Tip of the Week

QUESTIONS?
Board of Elders Meetings

- Hassan, Kyu Bin, Jeffrey (Miranda)
- Ruiqi (Ben S.)
- Hilery, Crystal (Ben A.)
- Jordan (Ben S.)
PLATFORMER4
PLAYTESTING!