CS 134

3D Graphics Optimizations

Today in Video Games

	Median		110%	110%
	Std. Deviation		5%	42%
	10th percentile		100%	0%

1000 more rows at bottom

		110%	110%
		5%	42%
		100%	0%

Next class thoughts...

How many people can bring
a laptop to next class?

Homework 3

- Controllable camera
- Camera must not be allowed to exit the world.

- Camera being controlled manually is fine.

Homework

- Many options for extra credit:
  - Do an isometric or hexgrid view
  - Add intelligent camera controls (character must not be always centered!)
    - Research on “camera controls with dead zone”

Optimization Mentality

- What makes Quicksort so fast anyway?

- If you tried to come up with a sorting algorithm, you would probably start with Selection Sort
  - Selection Sort O(n^2)
  - Quicksort O(n log n)

Optimization Mentality

- Selection Sort algorithm:
  - Look at each element of your array to find the minimal element.
  - Put the minimal element at the end of the sorted array
  - Repeat

- 4 8 5 6 7 2 1 3 9 0

Optimization Mentality

- Quicksort takes advantage of the transitive property of comparisons
- This extra insight is what makes quicksort fast

- Big-O notation is a measuring tool, NOT what makes the code fast.

Optimization Strategies

- Automated compiler optimizations
  - Generate better code!
  - Easy as flipping a switch
- Take better advantage of the hardware
  - The memory cache!
  - Data in native formats for GPU
- Reduce the amount of work done
  - Organize your data in cells / trees
  - Use geometric knowledge about your data

Optimizations

- What is the speed of drawing a game so far?
  - (d+a) * s + d*b Roughly O(n)
- Even though O(n) is good, drawing can be so slow that we want to dramatically reduce it!
- Be wary of anything that has to run every frame

Drawing Optimization #1

Release Mode
(Automated complier optimization)

Drawing Optimization #1

- Have the compiler generate better code for you.

- On Visual Studio you use “Release Mode”
- On GCC (Mac and Linux) -O2 or -O3

Drawing Optimization #1

TestSDL - Microsoft Visual Studio Express 2013 for Windows Desktop

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP

Local Windows Debugger | Debug | Win32

SDL_main.h | SDL.h | TestSDL.cpp | Solution Explorer

(Global Scope)

```
//// How to compile this on different platforms:
////
//// gcc TestSDL.cpp `pkg-config --cflags --libs sdl2 gl glew` -o TestSDL -lstdc++
#define SDL_MAIN_HANDLED
#include<SDL.h>
#include<GL/glew.h>
#include<stdio.h>
#include<string.h>
#include<assert.h>

/// Set this to true to force the game to exit
bool shouldExit = false;

/// The previous frame's keyboard state
unsigned char kbPrevState[SDL_NUM_SCANCODES] = { 0 };

/// The current frame's keyboard state
const unsigned char* kbState = NULL;
```

Drawing Optimization #1

- Visual Studio's Release Mode has its own set of configurations
- GCC -O3 contains optimizations that break under common buggy code.
  - -O2 is “safer”
- If you encounter bugs with optimizations turned on, it will be much harder to use a debugger
  - Assembly knowledge helps a lot here!

Drawing Optimization #2

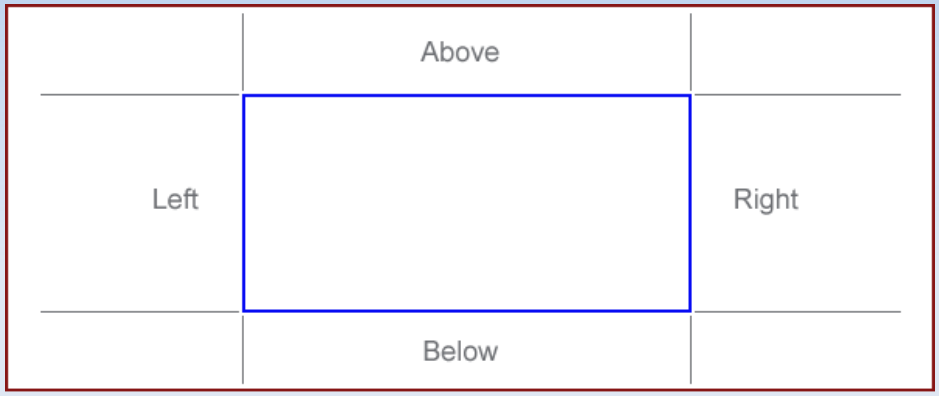
Don't Draw Offscreen Sprites
(Take better advantage of hardware)

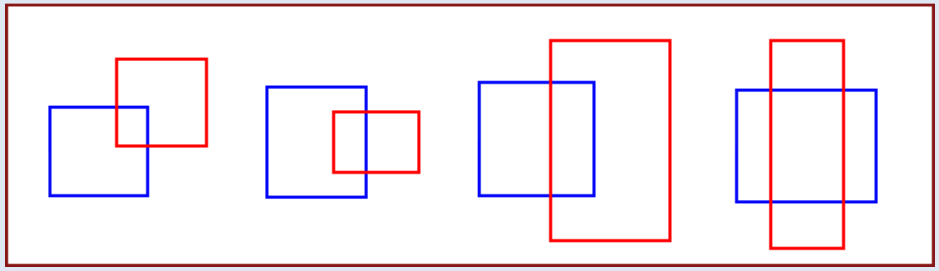
Drawing Optimization #2

- Current implementation sends all sprites and background tiles to the graphics card
  - Currently about 20,000 background tiles; 500 sprites
- Overhead from command based architecture
  - Every command takes memory
  - Every command uses the data bus CPU ↔ GPU
- On CPU, check if sprite is offscreen, if so, don't generate a command

Drawing Optimization #2

- Sprites, camera are both Axis-Aligned Bounding Box
- AABB/AABB test
  - If they don't intersect, then one box must be above, below, to the left, or to the right of the other box





Drawing Optimization #2

```
struct AABB {
    int x, y, w, h;
};
```

```
boolean AABBIntersect(AABB box1, AABB box2)
{
    // box1 to the right
    if (box1.x > box2.x + box2.w) {
        return false;
    }
    // box1 to the left
    if (box1.x + box1.w < box2.x) {
        return false;
    }
    // box1 below
    if (box1.y > box2.y + box2.h) {
        return false;
    }
    // box1 above
    if (box1.y + box1.h < box2.y) {
        return false;
    }
    return true;
}
```

Drawing Optimization #2

- Do AABB test BEFORE each draw

- if (AABBIntersect(camera, sprite))
  - DrawSprite(sprite)
- else
  - // Do Nothing

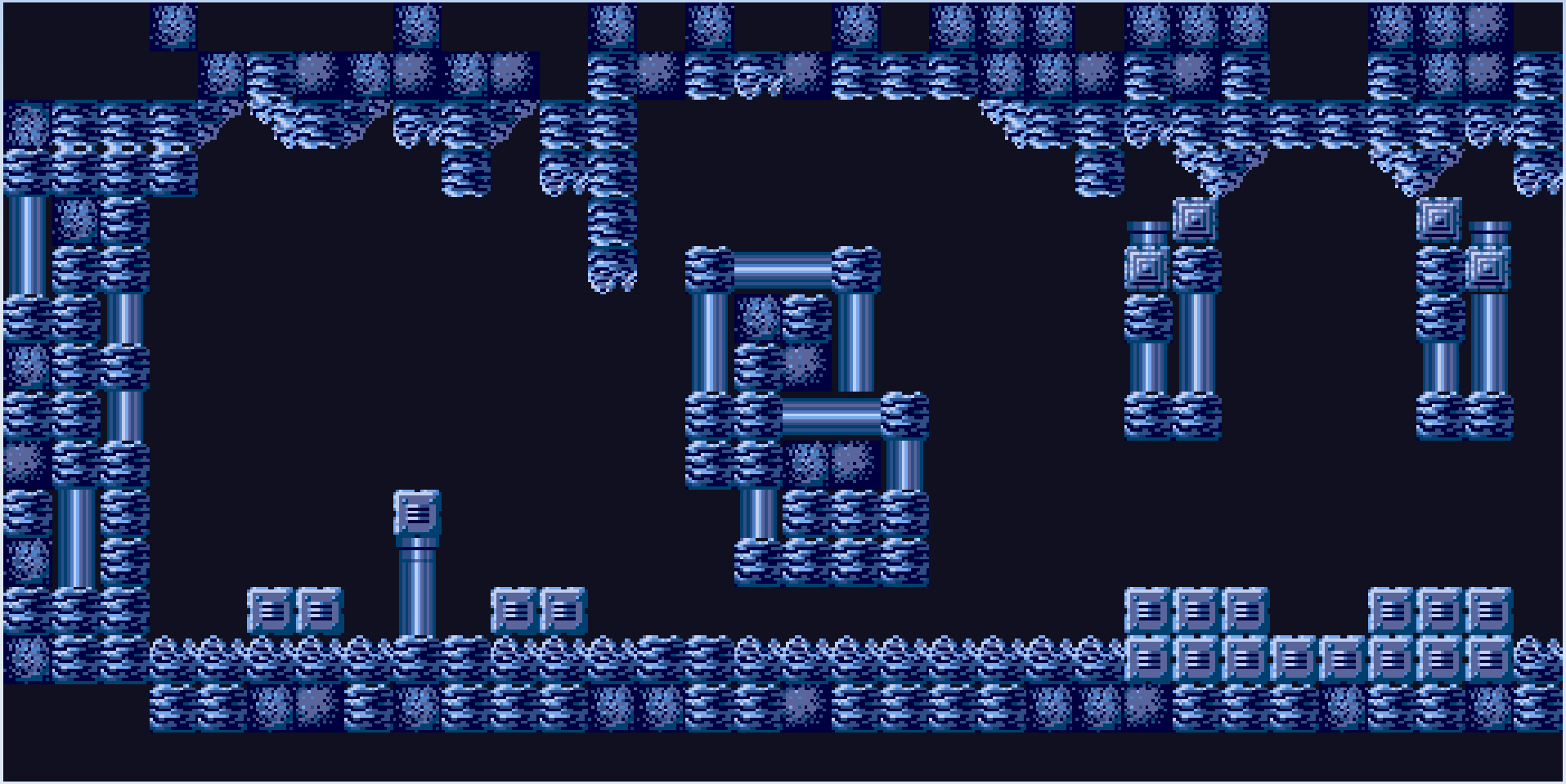
Drawing Optimization #3

Don't Process Backgrounds Offscreen At All
(Avoid doing unnecessary work)

Drawing Optimization #3

- We are still spending CPU time on every single background tile (all 20,000 or more)
- If we knew where to start and stop, we could avoid even looking at 90% of the tiles
- This does not need to be 100% accurate, it just needs to have no false negatives

Drawing Optimization #3



Drawing Optimization #3

- For grids:
  - tile_x = floor(camera_x / tile_width)
  - tile_y = floor(camera_y / tile_height)
- For hex-grids:
  - tile_x = floor(camera_x / tile_step)
  - tile_y = floor(camera_y / tile_step)
  - Bottom right coordinate needs extra offset!

Drawing Optimization #3

Drawing Optimization #4

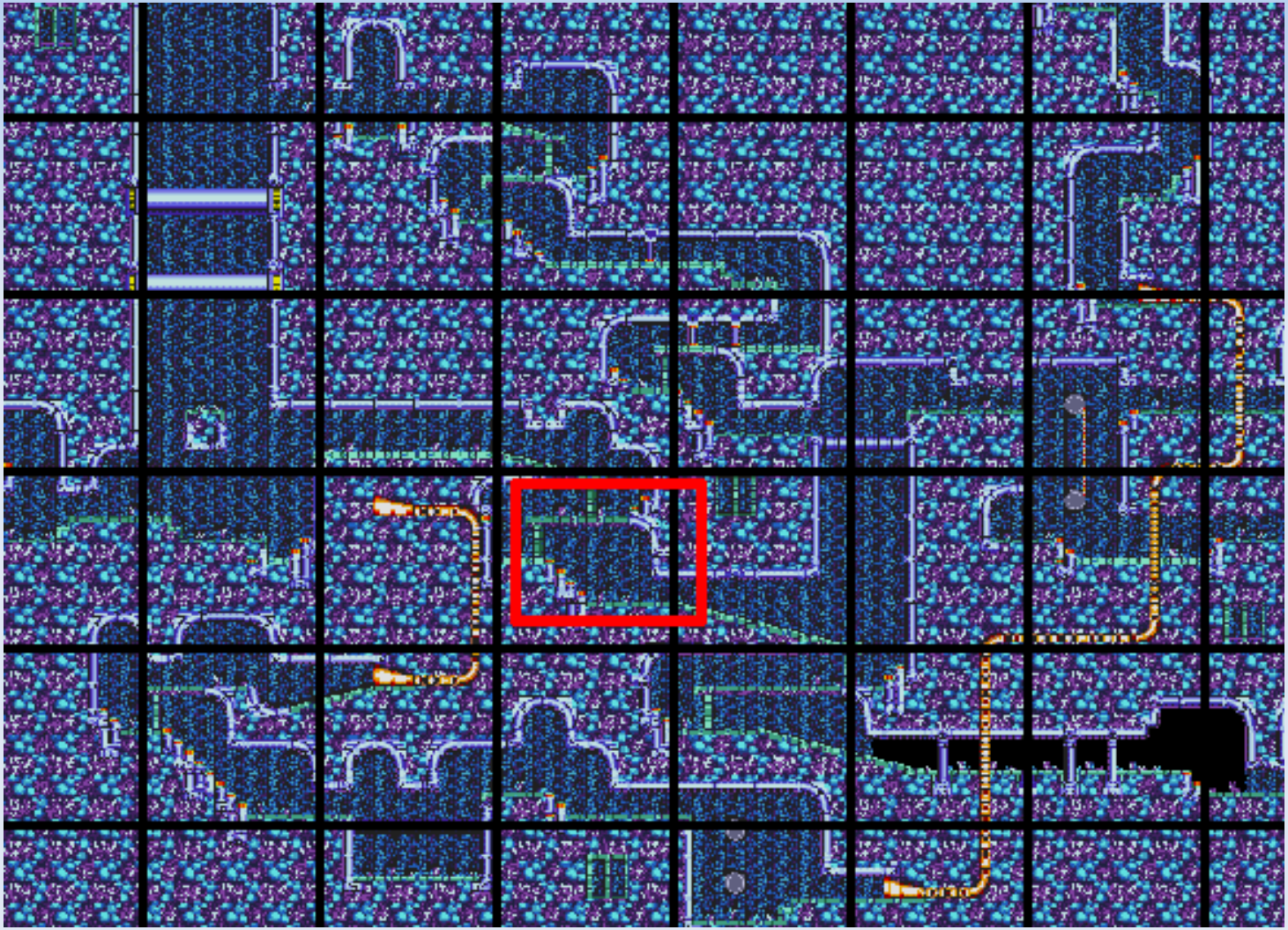
Do the same thing, for sprites
(Avoid doing unnecessary work)

Drawing Optimization #4

- Unlike backgrounds, sprites move, so the exact same optimization doesn't work
- Create logical buckets on a grid that the sprites can move between.

- This is similar to “Bucket Sort”

Drawing Optimization #4



Drawing Optimization #4

- Buckets should be big enough that there are at least tens of sprites per bucket
- Buckets should be small enough to exclude a significant percentage of the level
- During an update, you may need to re-bucketize the sprite

Drawing Optimization #5

Use the modern OpenGL API
(Make better use of the hardware)

Drawing Optimization #5

- The current implementation of DrawSprite() uses OpenGL “immediate mode”, from the early 90s.
- Extra slow!
  - No parallelism
  - Lots of extra copies on the CPU ↔ GPU
- There has been a better way since 2003, Vertex Buffer Objects (and Vertex Arrays in 1997)

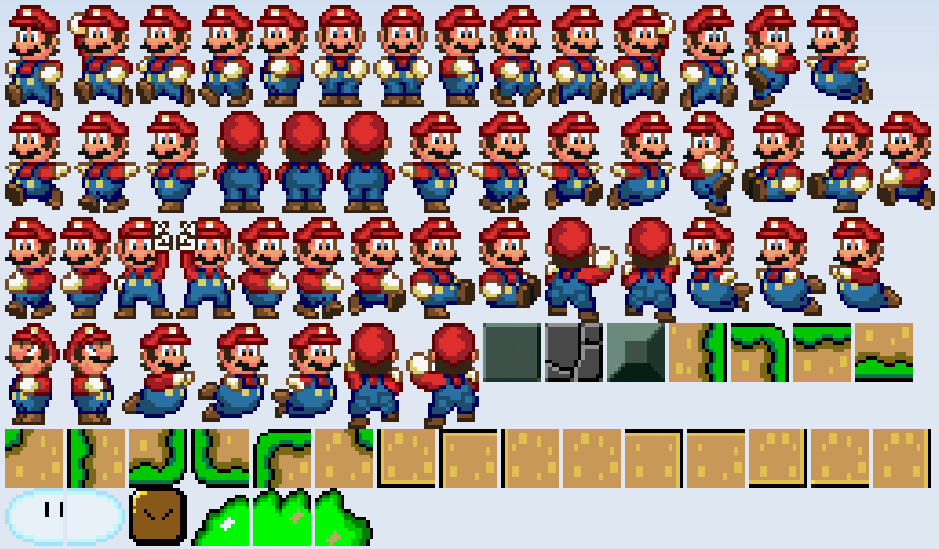
Drawing Optimization #5

- You build up a vertex buffer object each frame, copy it all over at the end, then draw it.
- glBufferData() - copies the data
  - Likely with usage=GL_STREAM_DRAW
- glDrawArrays() - draws the verts

- Note: This requires all the sprites drawn in one call to be in a single texture

Drawing Optimization #5

- Build a texture atlas
  - a.k.a. sprite sheet
- Draw subsets of the images using texture coords



Summary

- Release Mode
- Don't draw things that are off screen
- Only process background tiles that are on screen, make background a regular grid if necessary.
- Only process sprites that are near the screen, put them into a regular grid.
- Use the modern OpenGL interface.