Introducing the Final Project: The Fundamentals of Web Graphics

COS 426: Computer Graphics (Fall 2022)

Guðni Gunnarsson, Yuanqiao Lin, Yuting Yang

Agenda

- Final Project
  - Overview
  - Tips and Tricks
- Fundamentals of Web Graphics
- ThreeJS Crash Course
  - Anatomy of a Scene
  - A Basic Scene
  - Tips and Tricks

Final Project: Overview

- Timelines
  - Specifications now available
  - Proposals in-class, Dec 8 (1-2 slides)
  - Presentations and Demos, Dec 14 & 15
  - Written Report, Dec 16 (Dean's date)
- Work in groups of 2-4
  - Each group will be assigned to one TA after proposal due
  - Larger groups are held to higher expectations
  - Start forming groups NOW
- More details covered in Dec 1 lecture and project specs

Final Project: Expectations

- Finding an Idea
  - This project is truly open ended — pick any topic as long as your contributions are related to computer graphics.
  - Most groups will use ThreeJS (to which our advice will thus cater)
- Potential Pitfalls
  - Don’t overscope your project! Pick a well-defined idea that you can reasonably implement over a few weeks, set stretch goals
  - If going the ThreeJS route, avoid ideas that require complex external assets (e.g. large models and articulated animations)

Final Project: Expectations

- Implementation
  - You should **start reading tutorials early**, especially if you plan to work with ThreeJS. There are great guides out there, and we have linked some useful resources in the project specs.
  - You are allowed and encouraged to leverage online resources/libraries for your project
    - E.g. use a physics library so long as physics isn't your main focus
    - You may borrow from online tutorials and examples
    - **Don't let external resources dwarf your own code or contributions.** This has been a problem in the past.

Final Project: Tips & Examples

- Tips
  - Start early! Try to find a project group and project idea before the end of this weekend!
  - Reach out to TAs via Ed before the in-class proposal day to get extra feedback and advice
  - When brainstorming your project:
    - Look through projects on the ThreeJS homepage
    - Familiarize yourself with what ThreeJS can do via the library's official example page.
    - Check out the ThreeJS interactive editor (WIP)

Final Project: Tips & Examples

- Tips (continued)
  - Look through the Assignment 5 `coursejs/` scripts and try to understand how we are using ThreeJS.
  - Some simple starter code are available at the project specs to help you get started.
- Examples
  - Check out the final project “Hall of Fame” on our course site
  - View all of submissions from 2019 here
  - 2020, 2021

Fundamentals of Web Graphics

- The HTML Canvas
  - Everything you see (with one exception) is plain static HTML styled with CSS
    - Dynamic websites use JavaScript to interact with HTML
  - The one exception is the `<canvas />` element, which is a container for graphics drawn through JavaScript
  - Browser features like **WebGL** and JS libraries like **ThreeJS** allow application developers to draw complex scenes onto canvases.

Fundamentals of Web Graphics

← HTML Canvas

Fundamentals of Web Graphics

- Let's try animating a simple program, such as a cloth simulator
- Assume that the function already takes care of drawing what we want to the canvas
- **How do we go about animating our program?**

```
/*
 * A function that we want to
 * execute every frame
 */
const advanceProgram = () => {
    simulateProgram();
    drawProgram();
};

// Draw program once to screen
advanceProgram();
```

Fundamentals of Web Graphics

- **How do we go about animating our program?**
- Idea:
  - Use a while loop!

```
// Animation Attempt #1 (Bad)
while (true) {
    advanceProgram();
}
```

Fundamentals of Web Graphics

- **How do we go about animating our program?**
- Idea:
  - Use a while loop!
- Problem:
  - Will slow down your browser tab (JavaScript is not multithreaded)
  - What if your program is executing at an extremely high rate?

```
// Animation Attempt #1 (Bad)
while (true) {
    advanceProgram();
}
```

Fundamentals of Web Graphics

- **How do we go about animating our program?**
- Idea:
  - Use a callback to execute once every 60th of a second

```
// Animation Attempt #2 (Bad)

// Settings
const targetFPS = 60;
const msTimer = 1000/targetFPS;

// Invoke every msTimer millisecs
setInterval(
    advanceProgram,
    msTimer
);
```

Fundamentals of Web Graphics

- **How do we go about animating our program?**
- Idea:
  - Use a callback to execute once every 60th of a second
- Problem:
  - What if your program is slower than targetFPS?

```
// Animation Attempt #2 (Bad)

// Settings
const targetFPS = 60;
const msTimer = 1000/targetFPS;

// Invoke every msTimer millisecs
setInterval(
    advanceProgram,
    msTimer
);
```

Fundamentals of Web Graphics

- **How do we go about animating our program?**
- Idea:
  - Set timer from within animation loop

```
// Animation Attempt #3 (Bad)
const renderLoop = () => {
    advanceProgram();

    // Recur msTimer ms from now
    setTimer(
        renderLoop,
        msTimer
    );
};
renderLoop();
```

Fundamentals of Web Graphics

- **How do we go about animating our program?**
- Idea:
  - Set timer from within animation loop
- Problem:
  - What will happen to our framerate if our program takes a while?

```
// Animation Attempt #3 (Bad)
const renderLoop = () => {
    advanceProgram();

    // Recur msTimer ms from now
    setTimer(
        renderLoop,
        msTimer
    );
};
renderLoop();
```

Fundamentals of Web Graphics

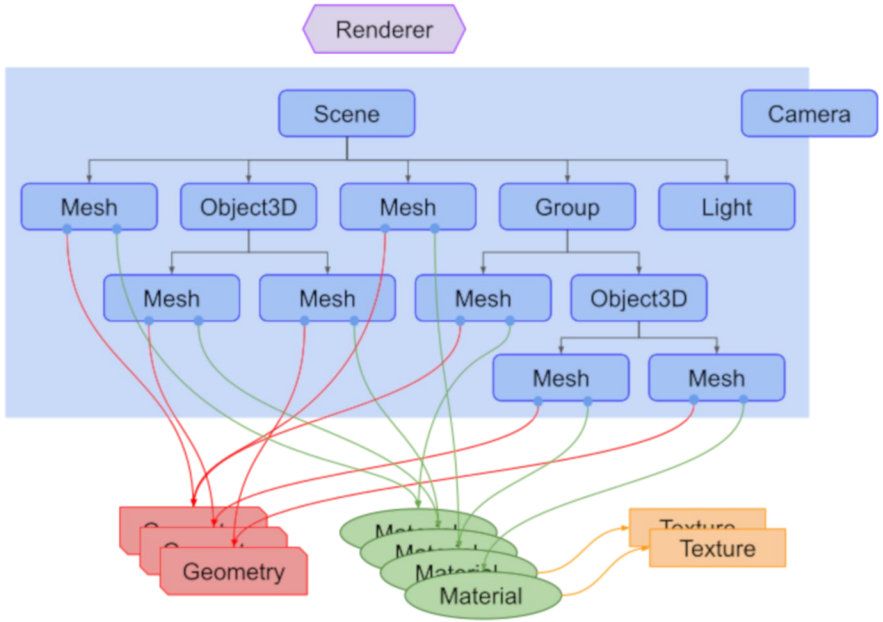
- **How do we go about animating our program?**
- Idea:
  - Use the optimized built-in function: `requestAnimationFrame`
  - Browser invokes callback before next repaint of screen

```
// Animation Attempt #4 (Correct)
const renderLoop = () => {
    requestAnimationFrame(
        renderLoop
    );
    advanceProgram();
};
// Invoke to start
requestAnimationFrame(
    renderLoop
);
```

ThreeJS: Anatomy of a Scene

A **Scene** is an organized collection of objects in space: meshes, cameras, and lights

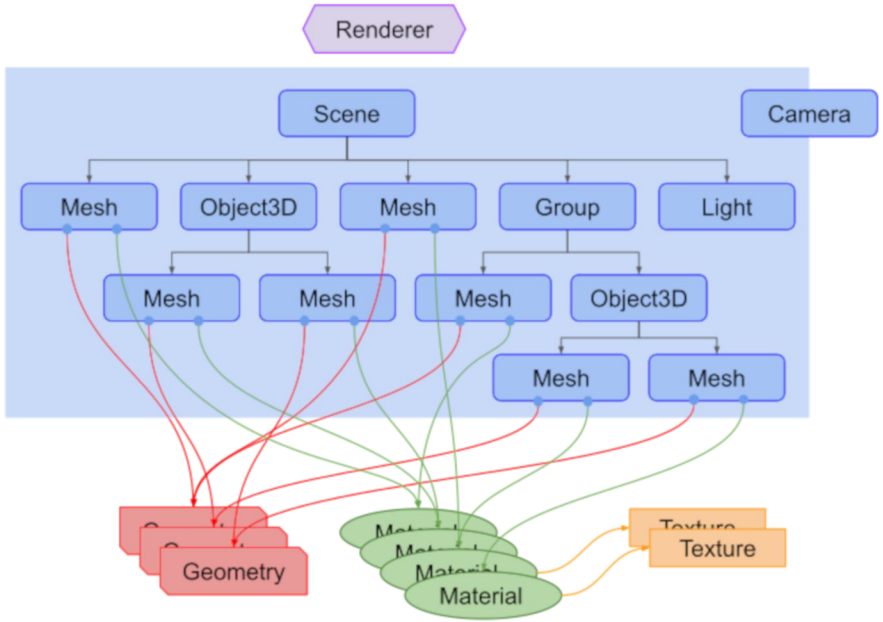
- **Meshes** consist of some polygonal **Geometry**, rendered using some visual **Material**
  - Materials can also use **Textures**
- **Cameras** let us view the scene
  - (from a particular position and angle)
- **Lights** illuminate the meshes in the scene, based on their material



ThreeJS: Anatomy of a Scene

Other useful scene objects:

- A **Renderer** draws a camera's view of the scene as pixels to the screen
- A **Group** is like a "sub-scene": lets us modify entire collections at once
  - Conceptually like a *reference frame* or *coordinate system*
  - Keeps your code clean!
- **Camera controls** alter a camera's params using keyboard + mouse
  - ThreeJS contains useful scripts to setup common control patterns



ThreeJS: A Basic Scene

- Set up a **renderer**, **scene**, and **camera**.
- Add **lights**.
- Create some meshes to populate the scene.
- Animate the scene.
- Make the scene **responsive**.

```
// Init scene
const scene = new THREE.Scene();
// Init camera (fov, aspect ratio, near, far)
const [width, height] = [
    window.innerWidth, window.innerHeight
];
const camera = new THREE.PerspectiveCamera(
    75, width/height, 0.1, 1000
);

// Init renderer; attach to HTML canvas
const renderer = new THREE.WebGLRenderer();
renderer.setSize(width, height);
document.body.appendChild(
    renderer.domElement
);
```

ThreeJS: A Basic Scene

- Set up a **renderer**, **scene**, and **camera**.
- Add **lights**.
- Create some **meshes** to populate the scene.
- Animate the scene.
- Make the scene **responsive**.

```
// Many lighting solutions in ThreeJS
// point lights, directional lights, etc.

const light = new THREE.HemisphereLight(
    0xffffbb,    // sky color
    0x080820,    // ground color
    1            // intensity
);

scene.add( light );
```

ThreeJS: A Basic Scene

- Set up a **renderer**, **scene**, and **camera**.
- Add **lights**.
- Create some **meshes** to populate the scene.
- **Animate** the scene.
- Make the scene **responsive**.

```
// width, height, depth
const cubeGeo = new
                    THREE.BoxGeometry(1,1,1);

const redMat = new
THREE.MeshPhongMaterial({color: 0xdd2244})

const c1 = new THREE.Mesh(cubeGeo, redMat)
c1.position.set(-2, 0, -5);
scene.add(c1);

const c2 = new THREE.Mesh(cubeGeo, redMat)
c2.position.set(+2, 0, -5);
scene.add(c2);

const cubes = [c1, c2];
```

ThreeJS: A Basic Scene

- Set up a **renderer**, **scene**, and **camera**.
- Add **lights**.
- Create some **meshes** to populate the scene.
- **Animate** the scene.
- Make the scene **responsive**.

```
// Animation Attempt #4 Adaption
const renderLoop = (timeMs) => {
     const time = timeMs * 0.0001;
     requestAnimationFrame(renderLoop);

     cubes.forEach((cube, index) => {
          const speed = 1 + index * 0.1;
          const rot = time * speed;
          cube.rotation.x = rot;
          cube.rotation.y = rot;
     });
     renderer.render(scene, camera);
};

// Set callback to begin animation
requestAnimationFrame(renderLoop);
```

ThreeJS: A Basic Scene

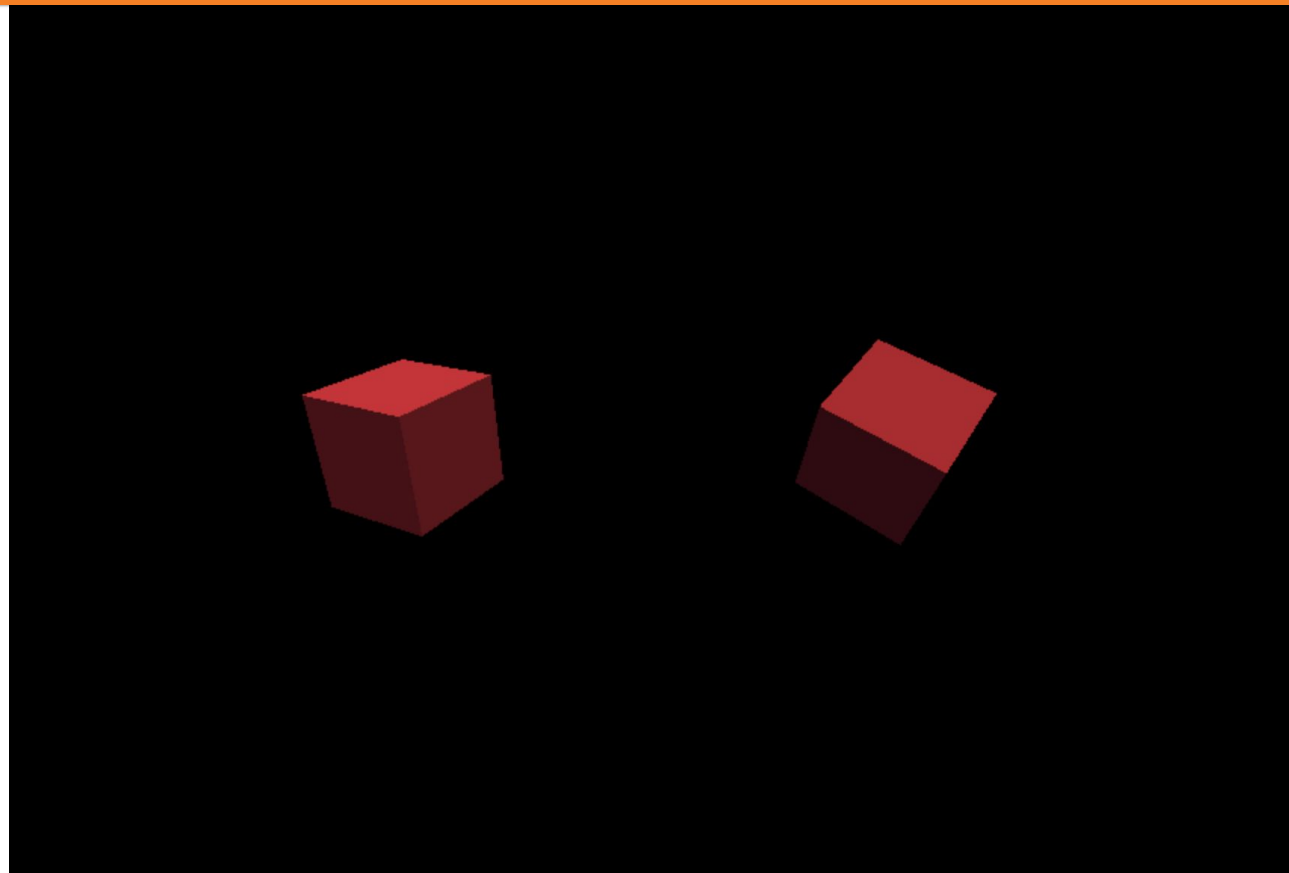
- Set up a **renderer**, **scene**, and **camera**.
- Add **lights**.
- Create some **meshes** to populate the scene.
- **Animate** the scene
- Make the scene **responsive**.

```
// Window resize event handler
const resizeHandler = () => {
     // Grab new width and heights
     const [width, height] = [
          window.innerWidth,
          window.innerHeight
     ];
     renderer.setSize(width, height);
     camera.aspect = width / height;
     camera.updateProjectionMatrix();
  }

// Add to resize event listener
window.addEventListener(
     "resize", resizeHandler, false
);
```

ThreeJS: A Basic Scene

Try the demo!



Final Project: Tips & Tricks

- For a project of this scope, the best projects are often stylized to take advantage of the simple geometries and materials ThreeJS provides off the bat.
  - For instance, using wireframe meshes and adding a final **bloom** pass will make everything look like a neon sign.
  - That said, ThreeJS also supports materials for very realistic textures.
- Consider using a physics engine, or look into using a **Web Worker** to take physics off the main thread if it's expensive.
- Familiarize yourself with the many **geometries**, **materials**, and **shaders** that ThreeJS provides!
  - The ThreeJS examples page has a good overview of these.

Final Project: Tips & Tricks

- Read the ThreeJS guide on how to **dispose of objects**.
- An **important optimization** is to **merge large objects** as described in this tutorial.
- Shallow Clone objects instead of recreating them
- Delegate certain tasks to each member of your group!
  - E.g. a single person should be responsible for physics, another for gameplay, yet another person for lighting, etc.
- Spend time planning out your project. A game-plan will save you time in the long run.
- Keep your code clean and well-organized!
- **Commenting** and **modularizing** your code from the start will only make your life (and your partners' lives) easier

Final Project: Tips & Tricks

- Try to use appropriate **abstractions** for your project whenever possible
  - e.g. an `Animal` class might have `Animal.move()`, `Animal.eat()`, `Animal.draw()`
  - Helps keep you focused on core logic, rather than boring details of updating & moving meshes
- Use **THREE.Group, THREE.Scene** (or the equivalent of your framework) to operate on groups rather than huge lists.
  - Related to this, take advantage of local reference frames within the global scene graph if you need coordinated movement