

COS 426 : Precept 11  
Particle Systems

# Agenda

- What you need to do
- Framework introduction
  - Updaters
  - Initializers
  - System Settings

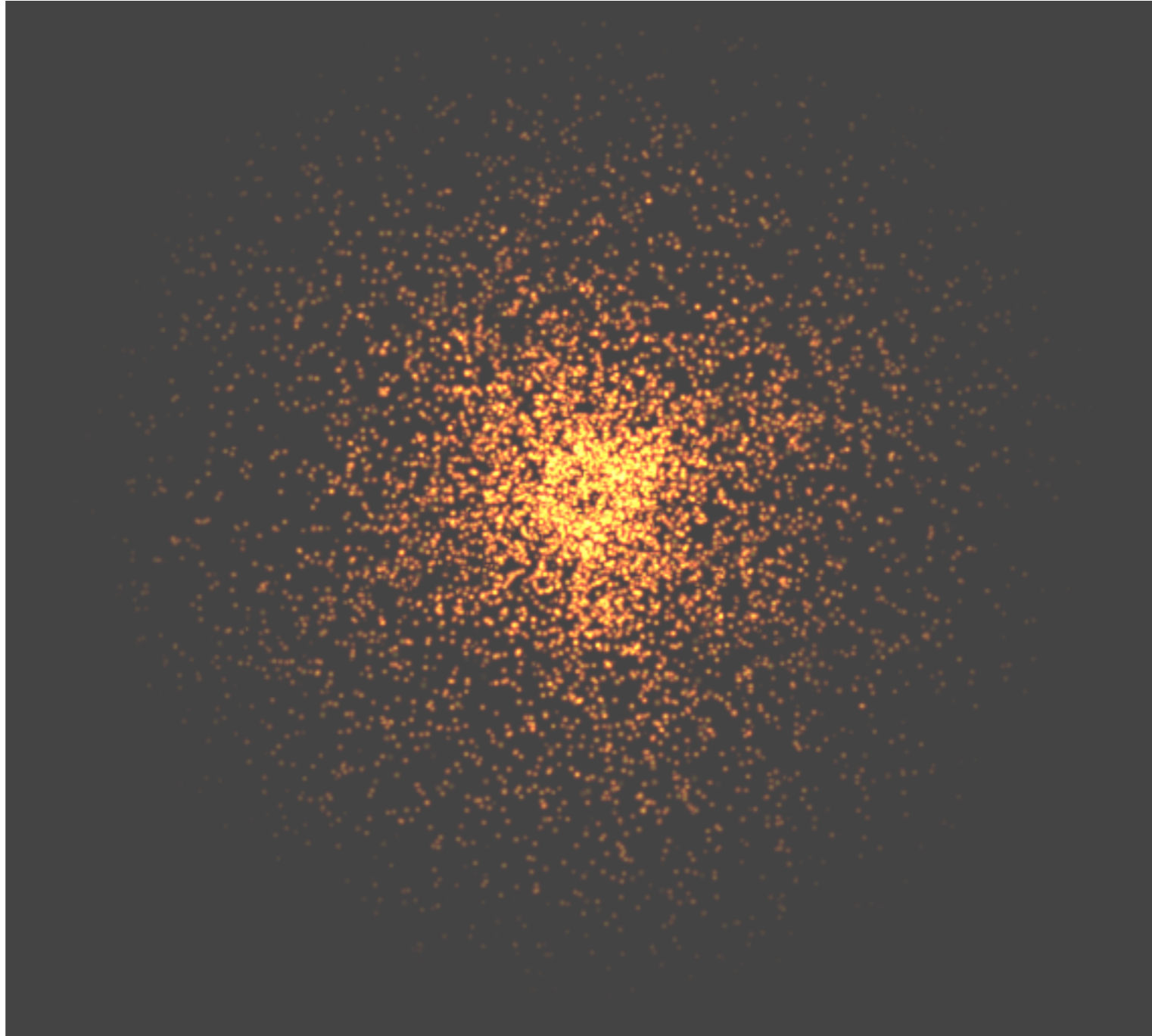
# Assignment details

- Easier, worth bit less points
- More freedom / burden on you
- Required features
  - Euler Integration
  - Sphere Initialization
  - Mesh Initialization
  - Simple collisions
  - Sinks
  - Cloth

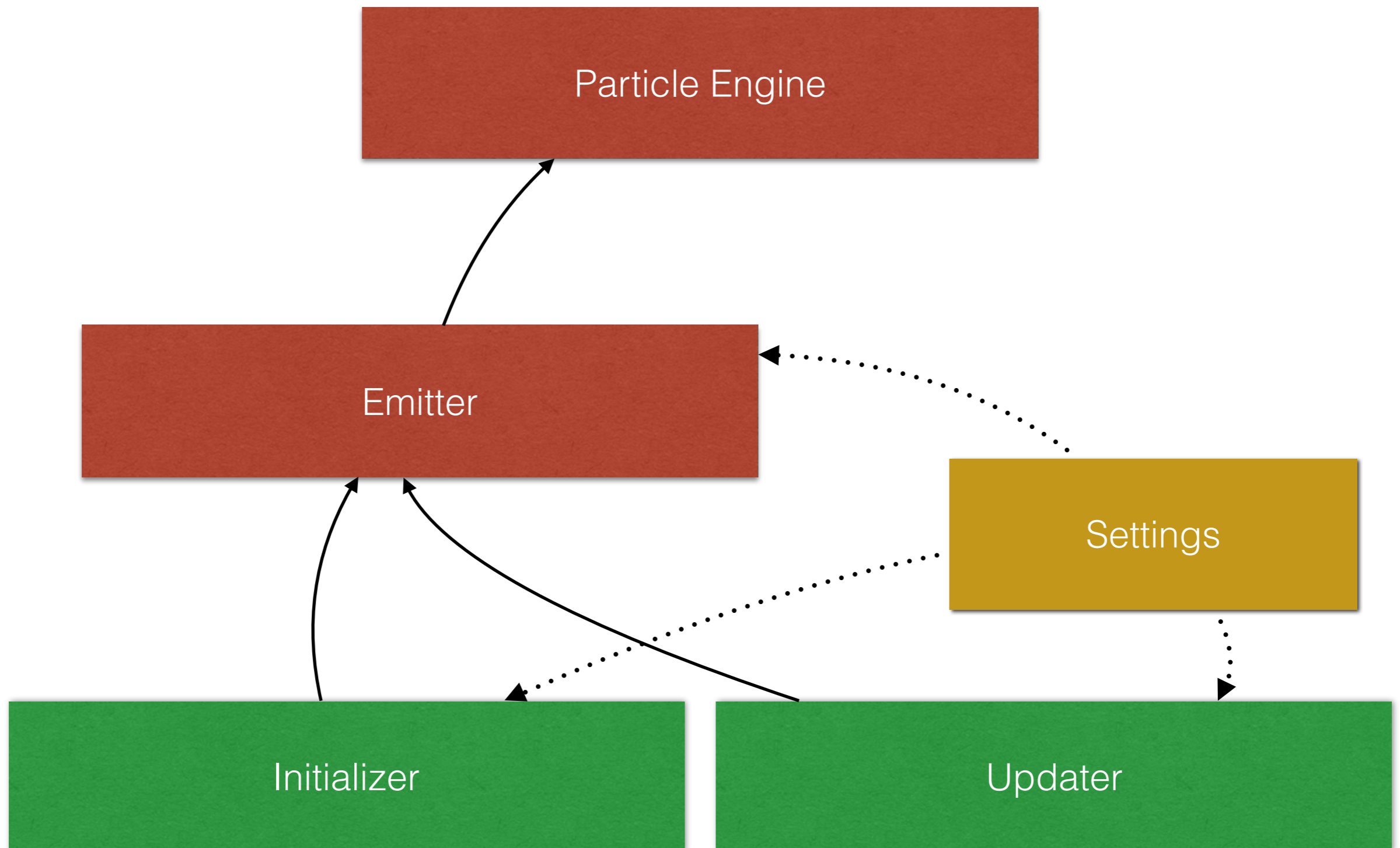
# Wait, slow down...

- Particle systems
  - Each particle has number of attributes
    - Positions
    - Velocities
    - Colors
    - Sizes
    - Lifetimes
    - Etc...
  - Initialize certain number each time-step
  - Update each at each time-step
- Different Initialization / update -> Different effects

Yay!



# Framework



# Initializer

- Function that specifies how new particles are generated
  - What is the position
  - What is initial velocity
  - Etc.
- Takes in a set of options
  - Can be anything that you need
  - Stored in this `._opts` variable
- Must implement initialize function!
  - Framework calls `MyInitializer.initialize()`;

# Initializer

- Example
  - `new MyInitializer( {position: new THREE.Vector3() } );`
  - Position can be accessed as
    - `this._opts.position`
  - `initialize ( particleAttributes, toSpawn )`
    - `particleAttributes` - arrays of positions, velocities, etc.
    - `toSpawn` - array of indices into these arrays
  - Particle Engine manages when to remove the particles
    - Just use `toSpawn` array!



# Arrays side note

- Fixed size buffer
  - Buffer stores max. 1000 particles
  - Generate 100 particles per second
  - Each particle lives 11 seconds
- Need to be able to know when particle is dead to free up space in array
  - Lifetime  $< 0$  — —  $>$  kill particle (utils.js)
- This is managed for you!
  - If you're interested we can talk past May 1st.

# Updater

- Function that specifies how new particles are updated
  - Apply forces to particles
  - Do collision detection
  - Change velocities/colors etc.
- Wide range of possible effects!
- Takes in set of options
  - Can be anything that you need
  - Stored in this.\_opts variable
- Must implement update function!
  - Framework calls `MyUpdater.update()`;

# Updater

- `new MyUpdater( {gravity: new THREE.Vector3(0, -10, 0) } );`
- Can be accessed as
  - `this._opts.gravity`
- Similarly you can pass other useful things:
  - Collidable objects
  - Sinks
- `initialize ( particleAttributes, alive, delta_t )`
  - `particleAttributes` - arrays of positions, velocities, etc.
  - `initialized` - array of specifying whether particle is initialized ( active )
    - Only update active particles
  - `delta_t` - global time, used for your integration

# Settings

- Particle Engine can be specified with set of settings
  - `systemSettings.js`
- Need to specify
  - Updater + Updater options
  - Initializer + Initializer options
  - Material
  - Max Particle Count ( Buffer Size )
  - Particle Frequency

# Settings

Sufficient for all required features



```
SystemSettings.mySystem = {  
  
    // Particle Material  
    particleMaterial : SystemSettings.standardMaterial,  
  
    // Initializer  
    initializerFunction : VoidInitializer,  
    initializerSettings : {},  
  
    // Updater  
    updaterFunction : VoidUpdater,  
    updaterSettings : {},  
  
    // Scene  
    maxParticles: 1000,  
    particlesFreq: 1000,  
    createScene : function () {},  
  
};
```

# Settings - Optional

- Scene
  - Just write a function that creates THREE.js objects
  - Add them to the scene

```
createScene : function () {  
  var sphere_geo = new THREE.SphereGeometry( 1.0, 32, 32 );  
  var phong      = new THREE.MeshPhongMaterial( {color: 0x444444,  
                                                emissive:0x442222,  
                                                side: THREE.DoubleSide } );  
  var sphere = new THREE.Mesh( sphere_geo, phong )  
  
  sphere.position.set (30.0, 30.0, 30.0);  
  Scene.addObject( sphere );  
},
```

# Settings - Optional

- Cloth
  - Rendered differently
  - Need to define grid structure

```
// Cloth specific settings  
cloth : true,  
width : 20,  
height : 20,
```

- max. particle count and particle frequency ignored.

# Utils.js

- particleAttributes arrays access
  - Huge array of numbers
    - `particleAttributes.position[0]` returns number, not `THREE.Vector3` !
  - Provided functions
    - `getElement( i, attribute )`
    - `setElement( i, attribute, val )`
    - `getGridElement( i, j, width, attribute )`
    - `setGridElement( i, j, width, attribute, val )`
- Also have function to kill particles
  - Useful for sinks!