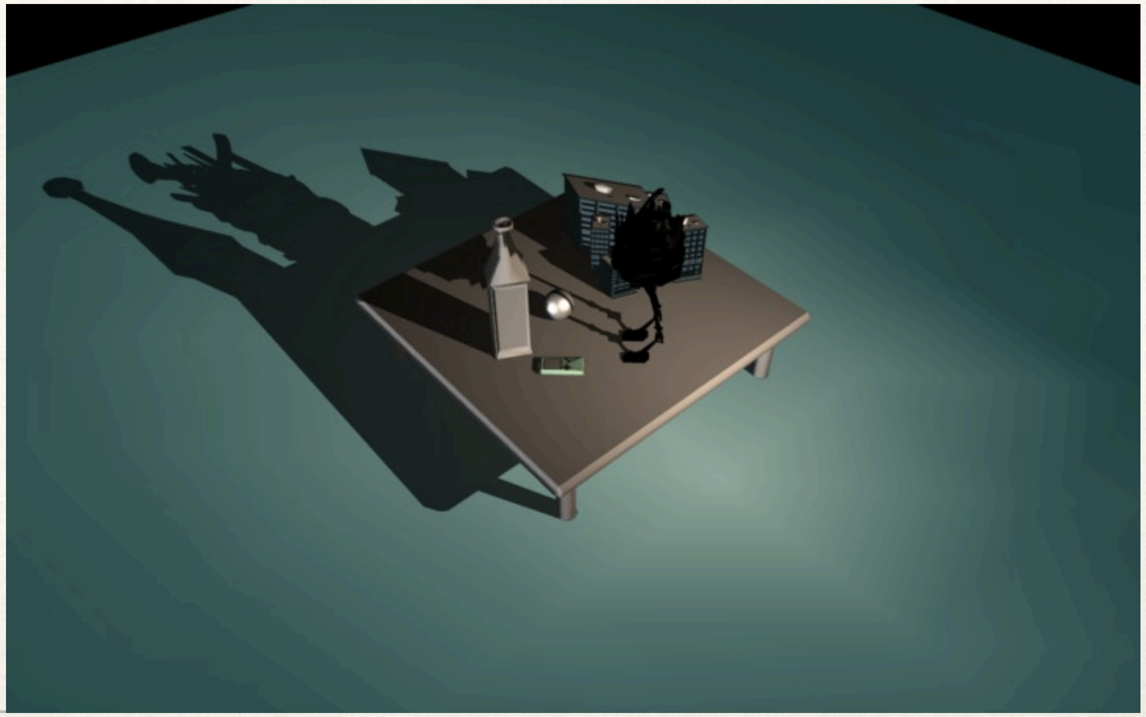


Dr. Sarah Abraham
University of Texas at Austin
Computer Science Department

Three.js Scenes

Elements of Graphics
CS324e

Object3D

- Base “class” for most objects in Three.js
    - Technically a prototype but you can think of it as something similar to a class!
- Provides properties and methods for working with scene objects
- Properties:
    - `.position`, `.scale`, `.rotation` represent local translation, scale, and rotation respectively
    - Can update using `set(Vector3 v)`: `object.position.set(15, 20, 0);`
    - Can update using `translateX`, `translateY`, `translateZ`, `rotateX`, `rotateY`, `rotateZ`

Objects in World Space

- World space is space at scene level
- `getWorldPosition(Vector3 v)`, `getWorldQuaternion(Quaternion q)`, `getWorldScale(Vector3 v)` return a vector/ quaternion or argument in world space
- Remember that local and world space are different systems once we begin working with scene hierarchies

Scene Hierarchies

- Objects can be added as children of other objects
  - `parentObject.add(childObject);`
- Objects can have one parent (`childObject.parent` returns an Object3D)
- Objects can have many children (`parentObject.children` returns an Array of Object3Ds)

Groups

- Similar functionality to adding child/parent objects via Objects3D, but makes hierarchy clearer

```
let object1 = new THREE.Mesh(mesh1, material1);

let object2 = new THREE.Mesh(mesh2, material2);

let group = new Group();

group.add(object1);

group.add(object2);

scene.add(group);

//group.children = [object1, object2]

//group.parent = scene
```

Math Functions

- Many different Math functions using Math-type objects
- Libraries for Vector2, Vector3 and Vector4 functionality
  - `add(Vector v), addScalar(Float s), angleTo(Vector v), dot(Vector v), length(), lerpVectors(Vector v1, Vector v2, Float alpha)`
- Libraries for Box, Sphere, Plane, Ray, Triangle functionality
  - Can check intersections, inclusion, distance to points etc
- Can also use Javascript Math library for basic trigonometric functions

Geometries

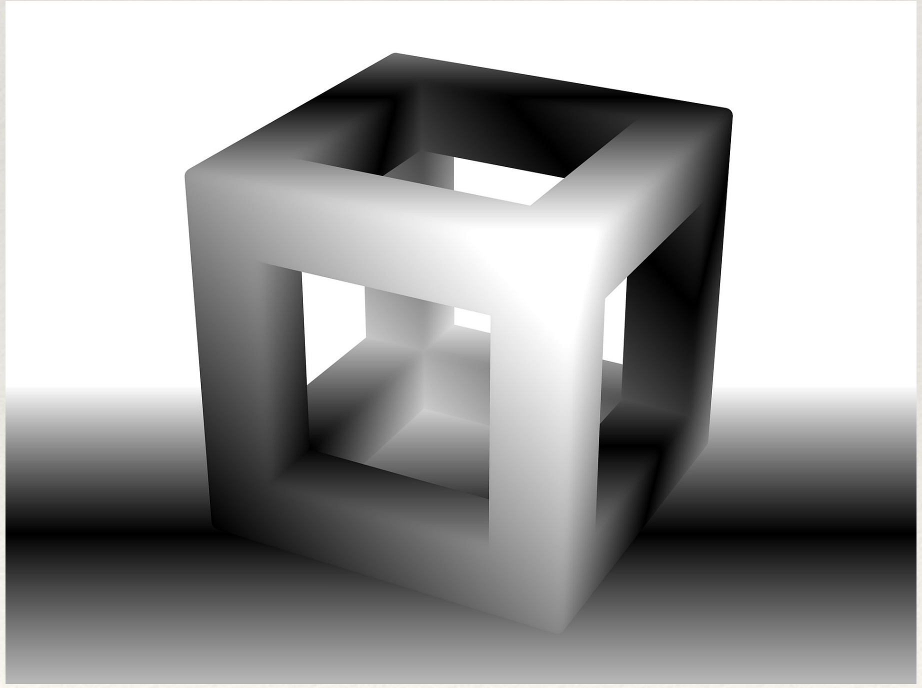
- BoxGeometry has width, height and depth as well as width, height, and depth segments
- SphereGeometry has radius, width and height segments
    - Spheres composed of triangles so number of segments determine smoothness of sphere
- CylinderGeometry has top radius, bottom radius, and height
- ConeGeometry has radius and height
- ShapeGeometry defined by an array of shapes (paths such as BezierCurves)

Materials

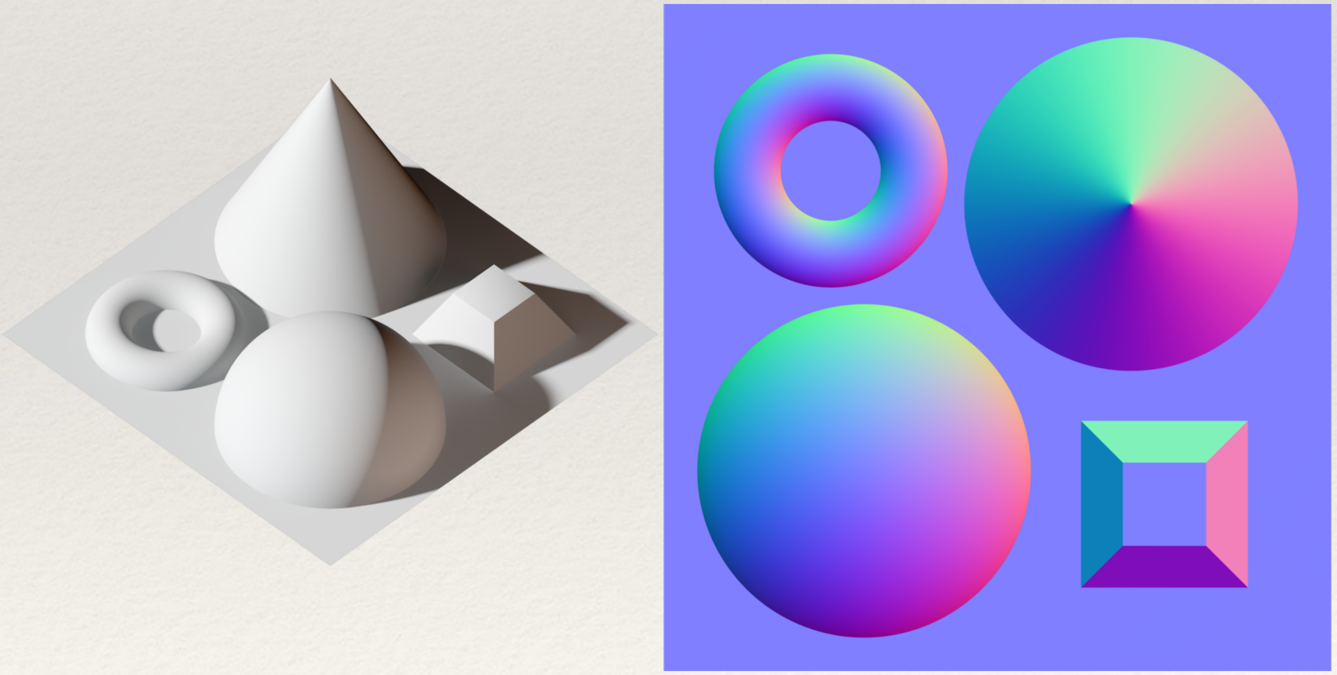
- MeshBasicMaterial has an ambient color but not affected by lights
- MeshPhongMaterial has Phong properties (ambient, diffuse, and specular properties)
    - `.color` (ambient), `.shininess` and `.specular` (specular), diffuse is built in
- MeshStandardMaterial has Phong properties as well as roughness, metalness and reflectivity
- Can apply environment maps to Phong and Standard materials using textures

Additional Materials

- Can also create Materials for more advanced mappings
- MeshDepthMaterial and MeshNormalMaterial allow for depth and normal mappings



Depth map



Objects plus their normal maps

Texture Mapping

- Texture maps can be loaded and applied to images via Material objects:
    1. Create a TextureLoader
    2. Load in an image as a texture and apply it to a material
    3. Apply the material to a mesh
- Phong and Standard Materials can include other types of maps that affect light on the material
    - `.alphaMap`, `.aoMap`, `.envMap`, `.normalMap`, `.roughnessMap` etc

Texture Mapping Example

```
let loader = new THREE.TextureLoader();

let texture = loader.load(‘path_to_image’);

let material = new MeshBasicMaterial({map:
texture});

…

let cube = new THREE.Mesh(geometry,
material);
```

Lights

- Basic lighting is supported:
  - AmbientLight has a color and intensity
  - DirectionalLight has a color, intensity, position and target (shines from position to target)
  - PointLight has color, intensity, position, distance and decay (determines how far the light shines and light falloff)
  - SpotLight has color, intensity, position, target, distance, decay, angle and penumbra
- `.castShadow` determines if non-ambient lights should cast shadows or not

Additional Lights

- HemisphereLight is positioned directly above the scene and shines a color fading from skycolor (`.color`) to `.groundcolor`
  - Provides more natural scene lighting
  - Does not support shadows
- RectAreaLight emits light from a rectangular plane
  - Has color, intensity, width, height, and lookAt (determines direction light is emitted)
  - Used for more realistic lights (also more expensive to compute)

Camera Controls

- OrbitControls provides basic functionality for positioning a camera within a scene:

  1. Include OrbitControls script from Three.js project file (examples->js->controls->OrbitControls.js) in current project directory
  2. Create OrbitControls
  3. Associate camera to OrbitControls
  4. Call update on OrbitControls object after any manual changes to the camera and/or in the draw loop if `.autoRotate` is set to true

OrbitControls Setup

```
<script src=“js/OrbitControls.js"></script>

…

let camera = new THREE.PerspectiveCamera(45,
window.innerWidth/window.innerHeight, 0.1,
1000);

let controls = new
THREE.OrbitControls(camera);

camera.position.set(0, 0, 20);

controls.update();
```

Key and Mouse Input

- OrbitalControls allows the camera to zoom, rotate, and pan
  - Zoom with mouse
  - Rotate with mouse right click
  - Pan with arrow keys
- Can control speed of controls with `.zoomSpeed`, `.rotateSpeed`, and `.panSpeed`
- Can set max and min values for zoom, rotate, and pan
- `.enableDamping` adds inertia to controls for better feel
  - Set `controls.enableDamping = true;`
  - Call `controls.update();` within draw loop

Hands On: Creating a Scene

- Extend the “Hello World” scene to contain the following:

1. Multiple objects
2. Multiple Phong materials
3. A directional and point light
4. A controllable camera