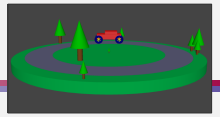
Hierarchical Modeling

- build subgraphs to represent complex objects

```
let tree = new THREE.Object3D();
let trunk = new THREE.Mesh(
    new THREE.CylinderGeometry(0.2,0.2,1,16,1),
    new THREE.MeshLambertMaterial({
        color: 0x885522
    })
);
trunk.position.y = 0.5;  // move base up to origin
let leaves = new THREE.Mesh(
    new THREE.ConeGeometry(.7,2,16,3),
    new THREE.MeshPhongMaterial({
        color: 0x00DD00,
        specular: 0x006000,
        shininess: 5
    })
);
leaves.position.y = 2;  // move bottom of cone to top of trunk
tree.add(trunk);
tree.add(leaves);
```

```
tree.position.set(-1.5,0,2);
tree.scale.set(0.7,0.7,0.7);
diskworldModel.add(tree.clone());
tree.position.set(-1,0,5.2);
tree.scale.set(0.25,0.25,0.25);
diskworldModel.add(tree.clone());
tree.position.set(5.1,0,0.5);
tree.scale.set(0.3,0.3,0.3);
diskworldModel.add(tree.clone());
tree.position.set(5.1,0,-0.5);
tree.scale.set(0.35,0.35,0.35);
diskworldModel.add(tree.clone());
tree.position.set(5.3,0,0);
tree.scale.set(0.5,0.5,0.5);
diskworldModel.add(tree.clone());
tree.position.set(-3.6,0,-3.6);
tree.scale.set(0.6,0.6,0.6);
diskworldModel.add(tree.clone());
tree.position.set(1,0,-5);
tree.scale.set(0.35,0.35,0.35);
diskworldModel.add(tree.clone());
```

the clone gets its own copy of the object's properties so changing the properties in the original object doesn't affect the clone



More About Transforms

- `Object3D` properties
  - `obj.scale`, `obj.rotation`, `obj.position` represent the scaling, rotation, and translation parts of the modeling transform
    - applied in that order
  - full modeling transformation applied to object is a combination of its `scale`, `rotation`, `position` combined with the modeling transformations applied to its ancestors in the scene graph

More About Transforms

- modifying transforms
  - modifies the `position`, `rotation` properties i.e. they apply in object coordinates, not world coordinates
  - `obj.translateX(dx)`, `obj.translateY(dy)`, `obj.translateZ(dz)`
  - `obj.translateOnAxis(axis,amount)`
    - `axis` must be a normalized `Vector3`
    - applies on top of previously-applied translations (object's `position`, `rotation` applied first)
  - `obj.rotateX(angle)`, `obj.rotateY(angle)`, `obj.rotateZ(angle)`
  - `obj.rotateOnAxis(axis,angle)`
    - `angle` in radians, `axis` must be a normalized `Vector3`
    - applies on top of any other rotations already applied (object's `position`, `rotation` applied first)

More About Transforms

- other operations
  - `obj.lookAt(vec)`
    - `vec` must be a `Vector3` in OC
      - if `obj` has no parent, OC = WC
    - rotates `obj` so that it faces `vec`, with up direction `obj.up`
      - default for `obj.up` is (0,1,0)

Lights

- types of lights
  - `intensity` is a non-negative brightness (> 1 allowed, but that's often too much light)
  - all have a `position` property
  - `new THREE.DirectionalLight(color,intensity)`
    - `position` property gives the direction from which the light shines (towards the origin)
  - `new THREE.PointLight(color,intensity,cutoff)`
    - `cutoff` is non-negative – 0 means illumination extends to infinity without attenuation, otherwise light falls off to 0 at cutoff distance
  - `new THREE.AmbientLight(color)`
    - color components should be small

Lights

- types of lights
  - `new THREE.SpotLight(color,intensity,cutoff, angle,penumbra,decay)`
    - `angle` is between 0 and π/2
    - `penumbra` is the percent of the spotlight cone that drops off to the edge of the cone (value 0-1, default 0)
    - `decay` is the amount the light drops off to the cutoff distance (default 2, use 0 for no attenuation)
    - `target` property is another scene graph node – light points at that position
      - default is a new empty `Object3D` at (0,0,0) whose position can be set
      - if target position is anything other than (0,0,0), target must be a node in the scene graph

```
spotlight = new THREE.SpotLight();
spotlight.position.set(0,0,5);
spotlight.target.position.set(2,2,0);
scene.add(spotlight);
scene.add(spotlight.target);
```

    - should use `MeshPhongMaterial` for objects illuminated by spotlights to get proper lighting effects

Positioning Lights

- lights are added to the scene graph like other objects
  - must be added to the scene graph to have an effect

- WC lights – fixed relative to the world
  - add light to root of the scene graph
- EC lights – fixed relative to the camera
  - add light as a child of the camera
  - add camera to the scene graph
- OC lights – fixed relative to an object
  - add light as a child of the object

Lights Example

```
// create some lights and add them to the scene.
// a light to shine in the direction the camera faces
let viewpointLight = new THREE.DirectionalLight("white", 0.8);
viewpointLight.position.set(0, 0, 1);  // shines down the z-axis
camera.add(viewpointLight);  // light in EC
scene.add(camera);
```

light attached to camera so its position is fixed relative to the camera – position in EC

```
// create some lights and add them to the scene.
scene.add(new THREE.DirectionalLight(0xffffff, 0.3)); // dim light shining from above
let viewpointLight = new THREE.DirectionalLight(0xffffff, 0.6);  // a light to shine in the direction the camera faces
viewpointLight.position.set(0, 0, 1);  // shines down the z-axis
scene.add(viewpointLight);
```

lights are attached to the top level of the scene

these are technically WC lights – "viewpoint" light acts as a viewpoint light because in this scene rotating the view is done by applying a modeling transform to the model rather than adjusting the camera

Image Textures

- steps
  - load the image for the texture
    - setting for proper color

```
var loader = new THREE.TextureLoader();
var texture = loader.load( imageURL );
```

```
texture.colorSpace = THREE.SRGBColorSpace;
```

  - add the texture to the object's material
    - set material's `map` property

```
material.map = texture;
```

    - BasicMeshMaterial, `LambertMeshMaterial`, `PhongMeshMaterial` can all use textures
      - texture color modulates the material color – set material color to white to use the texture's color as is
  - texture coordinates are part of the geometry
    - included with standard mesh geometries
- challenges
  - image loading is asynchronous, so scene must be redrawn once texture images are loaded

```
var texture = new THREE.TextureLoader().load( "brick.png", render );
```

this syntax illustrates *function chaining* – it's equivalent to the two steps for loading given above but is more compact and doesn't require naming the `TextureLoader` as a separate object

supply a callback function to draw scene as a parameter for `load()`

Texture Properties

- `Texture` object properties
  - `wrapS`, `wrapT`
    - default is `THREE.ClampToEdgeWrapping`
    - also `THREE.RepeatWrapping`, `THREE.MirroredRepeatWrapping`
  - `repeat`, `offset`, `rotation`
    - scaling, translation, and rotation portions of the texture transform, respectively
      - `repeat`, `offset` are of type `THREE.Vector2`
      - `rotation` is radians
    - applied to texture coordinates rather than the texture image, so the visual effect is the opposite of what is specified
      - e.g. `repeat` of 2 makes the texture image appear half the size