

# CSC 240 Computer Graphics

## Video 11: Three.js & WebGL

Nick Howe  
Smith College

*Some slides & content courtesy Sara Mathieson*

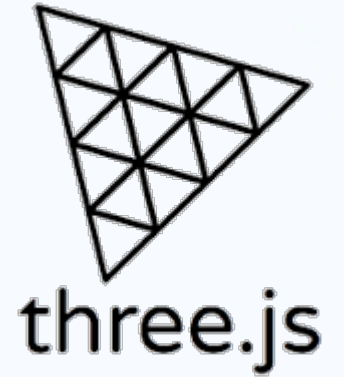
# WebGL

- Subset of OpenGL adapted for use in browsers\*
- OpenGL: “cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics”
- Common baseline standard for graphics programming



\* GL = Graphics Library

# Three.js



- Useful and popular library for WebGL: <https://threejs.org/>
- Allows us to create and manipulate 3D objects
- Still have a canvas (what we're drawing on/ in)



- Main new elements:
  - Scene: where we add 3D objects and lights
  - Camera: where our “eye” is, not part of the scene
  - Renderer: tool to draw the scene on the screen



# Scene



The scene object keeps track of all objects that will make up your output image

- For now, all we do with scenes is create them and add objects to them

```
// Create a new scene for adding objects to:  
scene = new THREE.Scene();
```

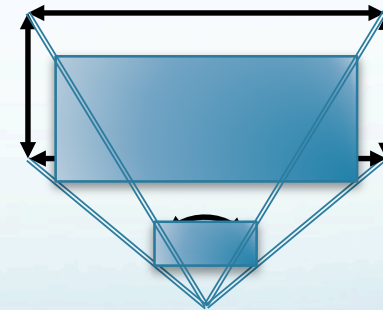
```
// Add an already-created object to the scene:  
scene.add(myObject);
```

# Camera



A camera object describes the projection type and viewing frustum to be used by the renderer to create the image

- It is positioned using world coordinates but is not part of the scene (invisible!)
- Specify four parameters to create perspective camera
  - Field of view (in degrees)
  - Aspect ratio (should match the canvas)
  - Near clipping plane
  - Far clipping plane

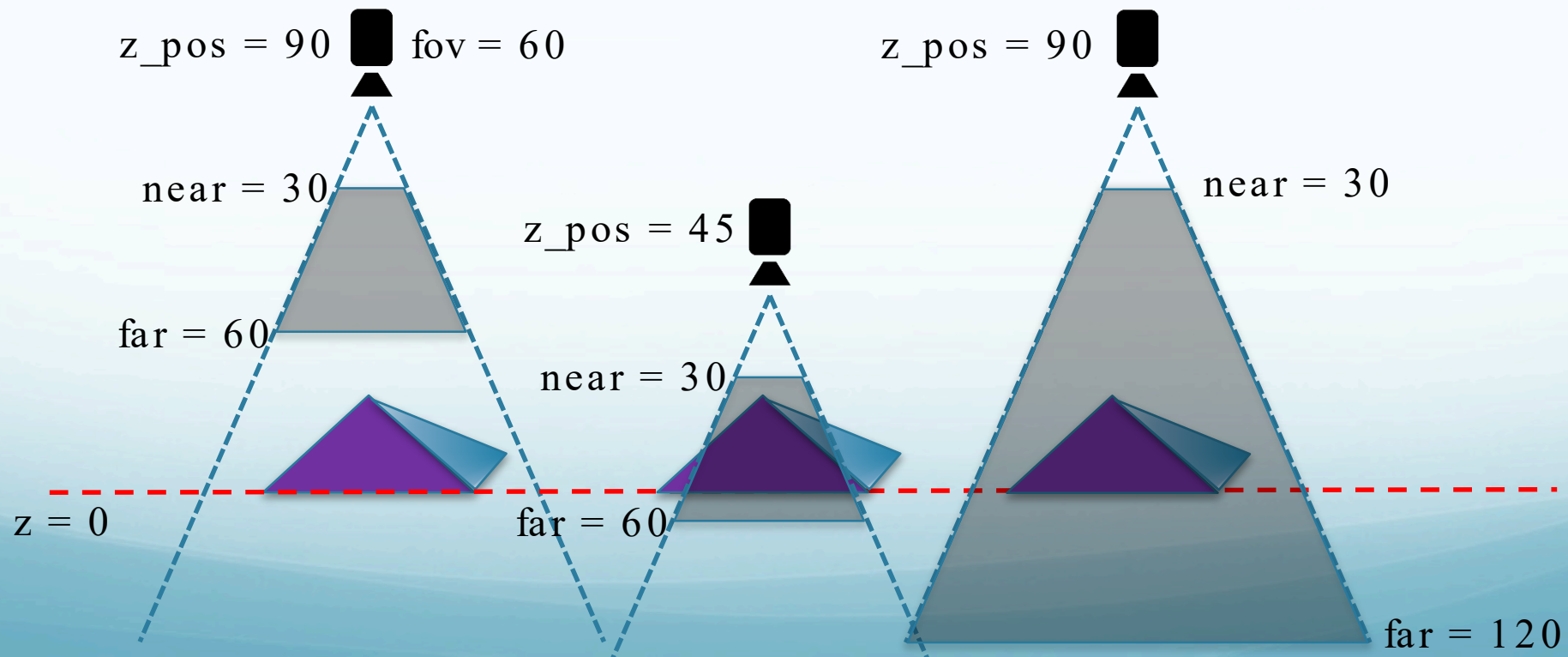


```
camera = new THREE.PerspectiveCamera(fov, aspect, near, far);  
camera.position.z = z_pos; // adjust position
```

# Choosing Camera Parameters

Camera parameters exert critical control over what is visible

- Position: **z\_pos** variable specifies height of camera above the world origin
- Clipping: **near** and **far** variables specify distances from the camera
- Field of view: **fov** variable specifies width of visible cone





# Renderer



The renderer does a lot of work behind the scenes

- Can take optional parameter list during creation

```
canvas = document.getElementById("glcanvas");  
renderer = new THREE.WebGLRenderer( { canvas: canvas, antialias: true} );
```

- Use it to set the background color

```
renderer.setClearColor(0); /// Set background color (0, or 0x000000, is black).
```

- Tell it to draw the scene when ready

```
/// Render the scene. This is called for each frame of the animation.  
function render() {  
    renderer.render(scene, camera);  
}
```

# Questions

PAUSE NOW & ANSWER

1. What is the relationship between OpenGL, WebGL, and Three.js?

*OpenGL is a generic graphics standard, WebGL is a variant specialized for web pages, and Three.js is an implementation and API for WebGL*

2. Where can I find documentation of Three.js? <https://threejs.org/>

3. Which object would be responsible for the following?

a. Changing the background color *Renderer*


b. Determining which objects are visible *Camera*

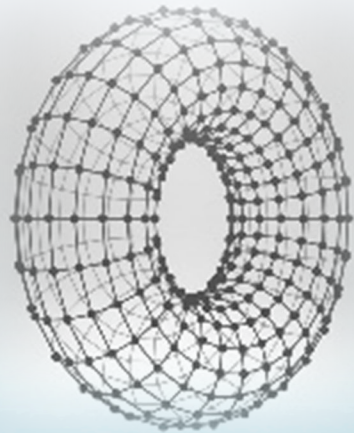
c. Placing a new object in relation to others *Scene*



# Scene Objects

Once your scene is ready, you'll want to fill it with cool looking stuff.  
What can go in a scene?

- **Lights** 
- **Meshes**, composed of
  - Geometry
  - Material



<https://cgabrieldesign.com/portfolio/dirty-rotten-scoundrels-theater-set/>  
VectorStock.com

# Lights

Lights illuminate the scene. No lights = you see nothing!

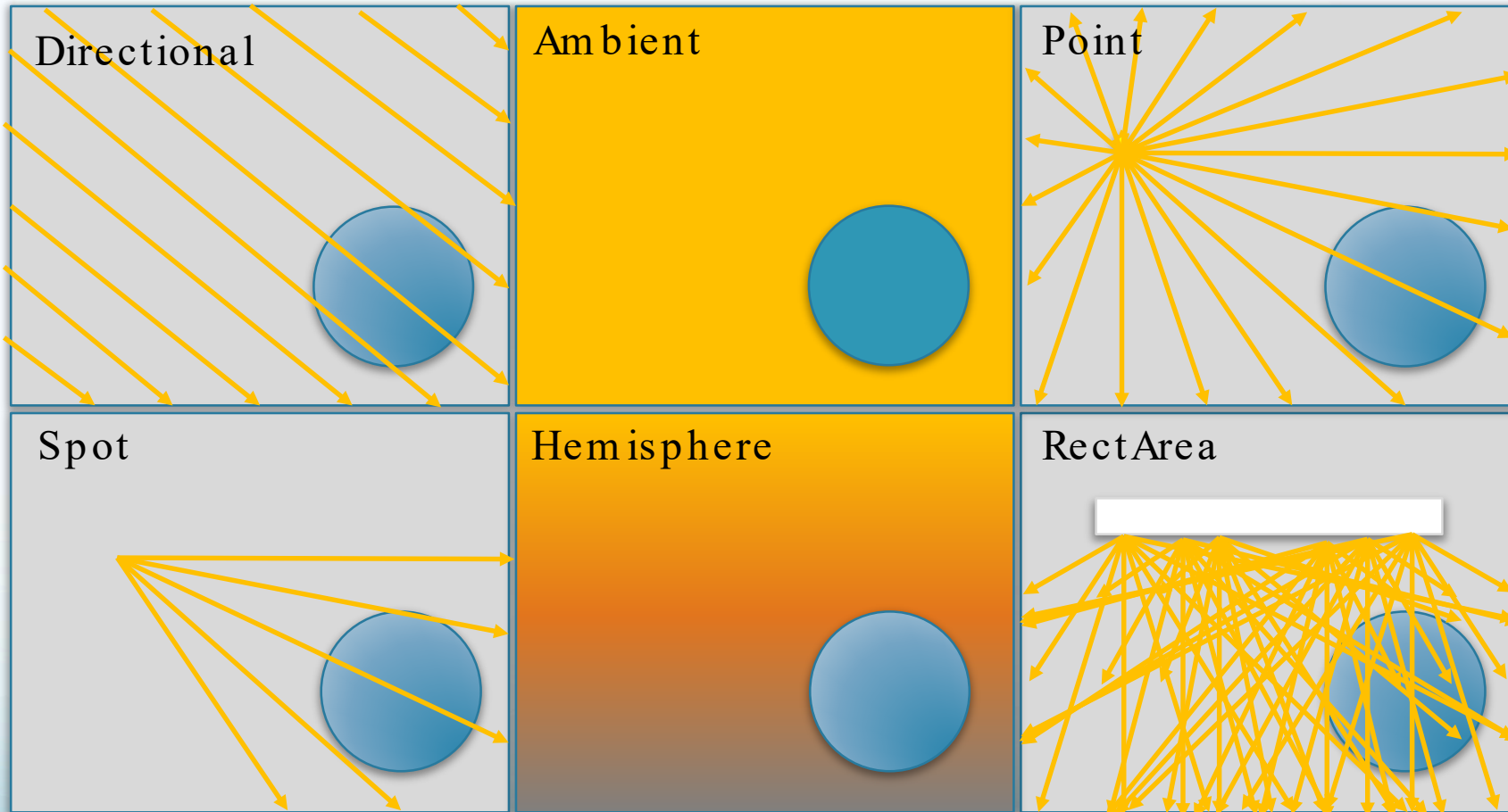
- Lights come in various configurations
  - Point sources
  - Rectangular emitters
  - Ambient light & hemisphere light
  - Directional & spot lights
- All lights have color and intensity
- Spatial parameters will vary by type of light

What might each of these be useful for?



```
// create bright white ambient light  
var ambientLight = new THREE.AmbientLight(0xffffffff,1);  
scene.add(ambientLight);
```

# Types of Lights in Three.js



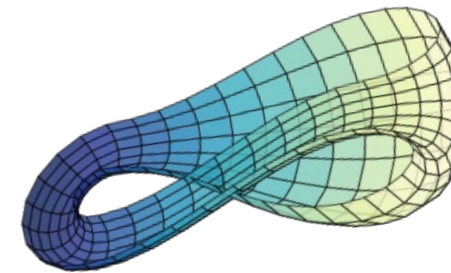
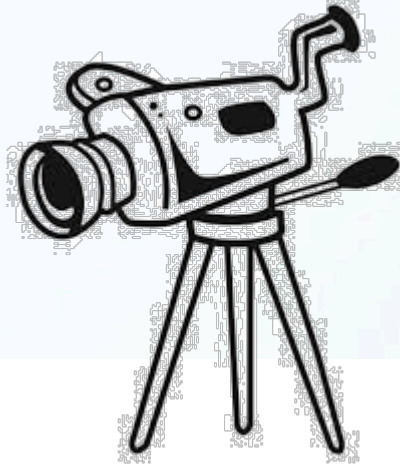
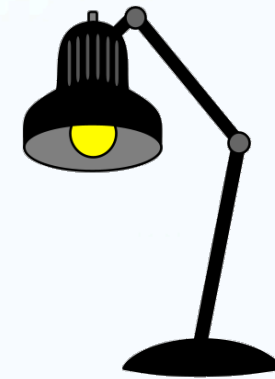
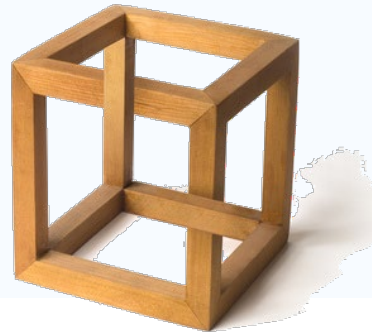


# Activity

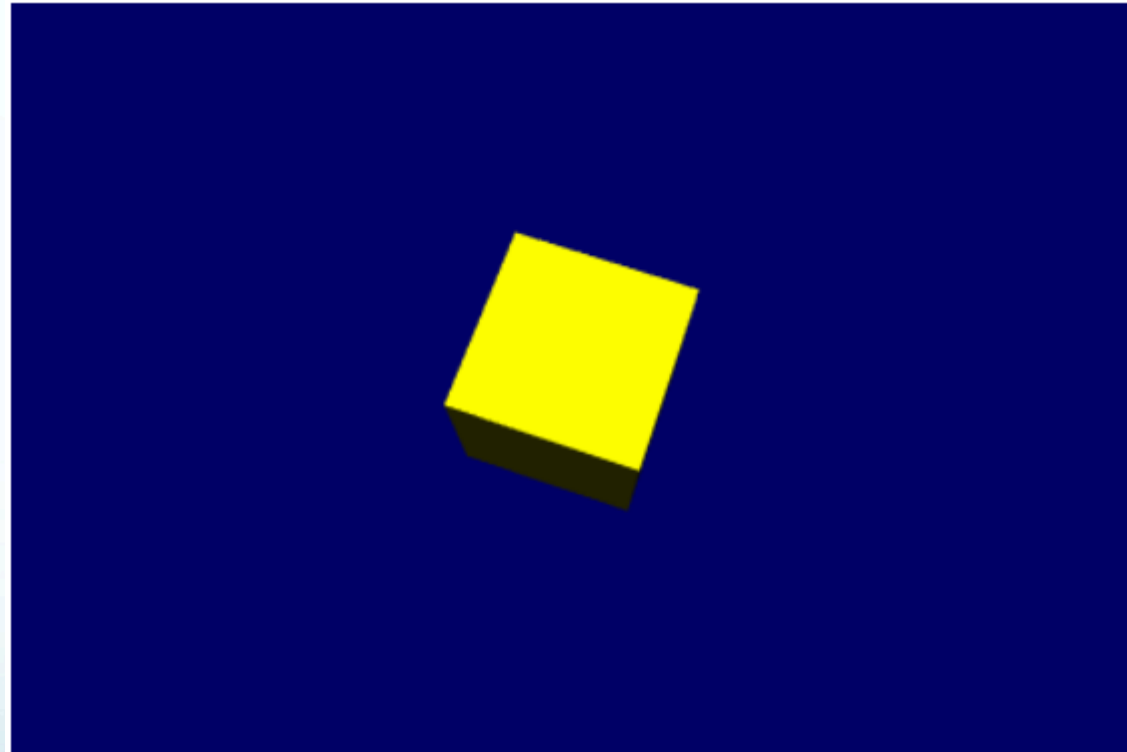
PAUSE VIDEO & RESPOND

Find the [three.js documentation web pages](#) and identify a class that represents each of the following:

- A type of light
- A type of camera
- A type of surface material
- The geometry for some predefined shape



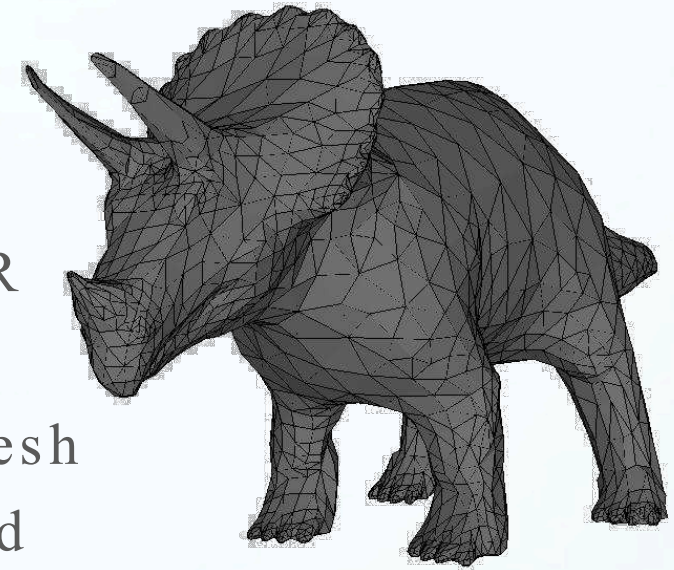
# Three.js Coding Demo



# Object Creation Checklist

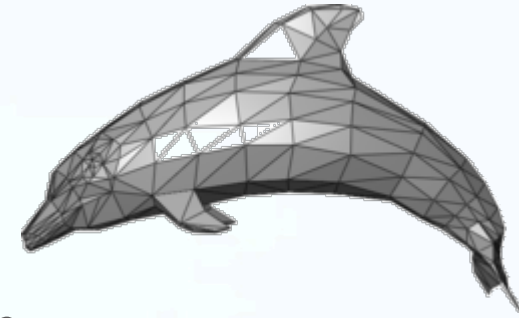
Steps to create a custom object in the scene:

1. Create a list of vertices, giving 3D coordinates
2. Create a list of faces, giving 3 vertices each using RHR
3. Specify materials for each face
4. Combine the geometry and materials together as a mesh
5. Add to scene and modify position & rotation as desired





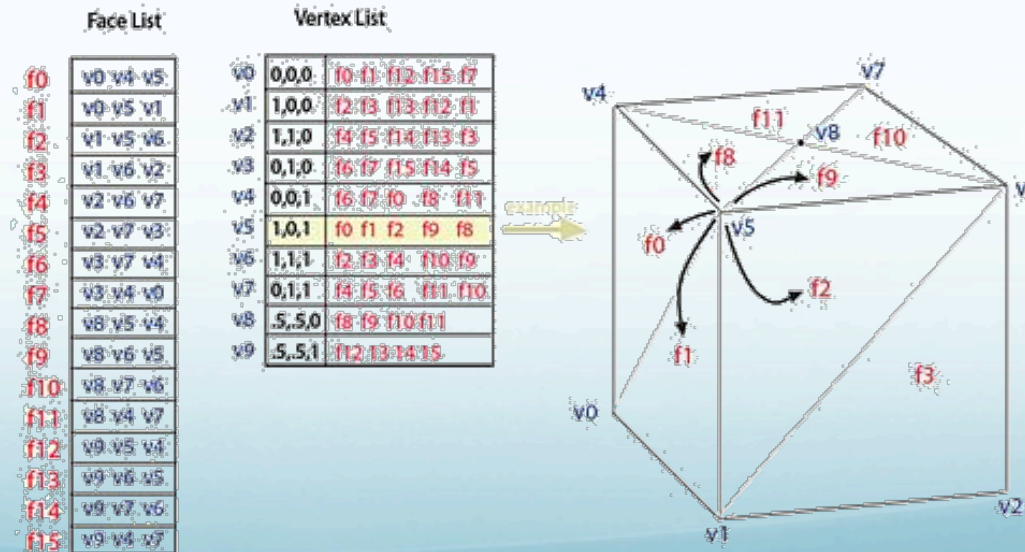
# Geometry



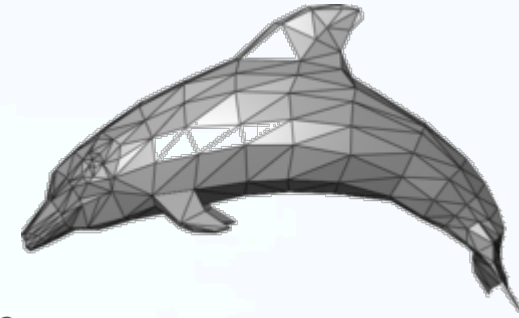
Geometry defines the positions of an object's surfaces

- **Vertices** are the corners
- **Faces** (made of triangles) connect coplanar vertices

Face-Vertex Meshes



# Geometry



Geometry defines the positions of an object's surfaces

```
/// set up the geometry for a shape
var myGeom = new THREE.Geometry();

/// create the (x,y,z) points needed for this shape
myGeom.vertices = [ /// array of Vector3 giving vertex coordinates
    new THREE.Vector3(1, 1, 0), /// vertex number 0
    /// ...
];

/// using the indices of the vertices above, create triangular faces
myGeom.faces = [ /// array of Face3 giving the triangular faces
    new THREE.Face3(0, 1, 2),
    /// ...
];
```

Order here matters!

# Geometry

Unlike real-world objects, in Three.js faces point only in one direction

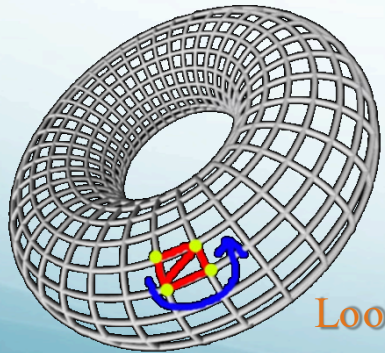
Reverse side  
is totally  
transparent!



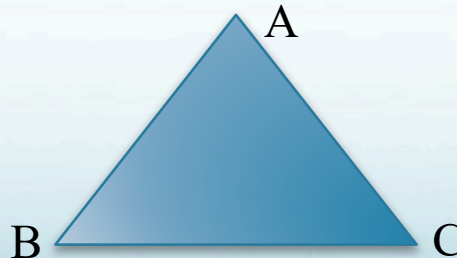
Actually,  
you won't  
even see  
the outline



Visible side is determined by **right hand rule**



Looking at the side you  
want visible, list the  
points counterclockwise.

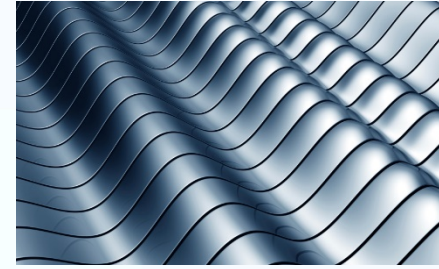


*Right hand rule:  
Fingers curl in order vertices are listed  
Thumb side is exterior*

A-B-C points out of screen (also B-C-A or C-A B)  
A-C-B points into screen (also C-B-A or B-A-C)



# Material



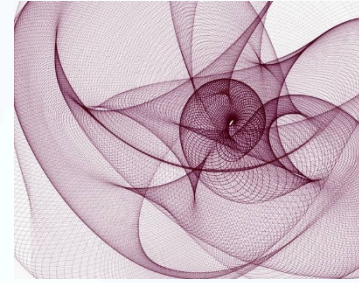
Materials define the way an object interacts with light in the environment (reflection, absorption, color mixing, etc.)

- Basic materials include color and reflectivity profile
- Advanced materials can have designs, texture, etc. – more on this later

```
// Creates a material for an object
// that is "matte" not "shiny"
var myMaterial = [
  new THREE.MeshLambertMaterial( {
    color: 0xffffffff, flatShading: true } ),
  // ...
];
```

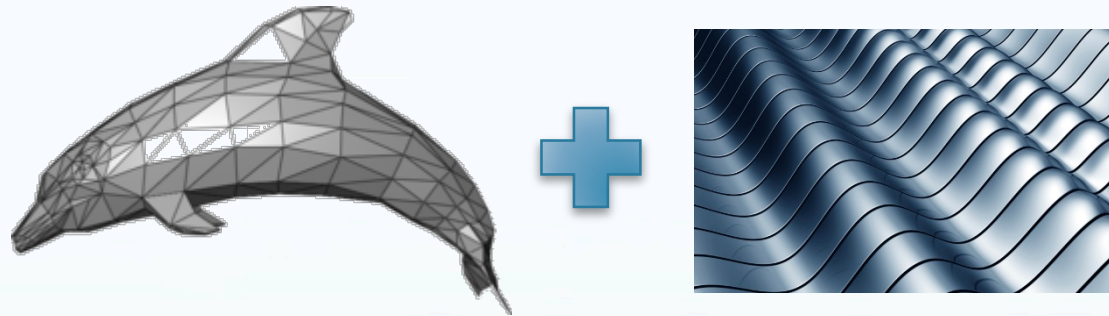
*MeshBasicMaterial*  
*MeshDepthMaterial*  
*MeshLambertMaterial*  
*MeshNormalMaterial*  
*MeshPhongMaterial*  
*MeshPhysicalMaterial*  
*MeshStandardMaterial*  
*MeshToonMaterial*

# Mesh



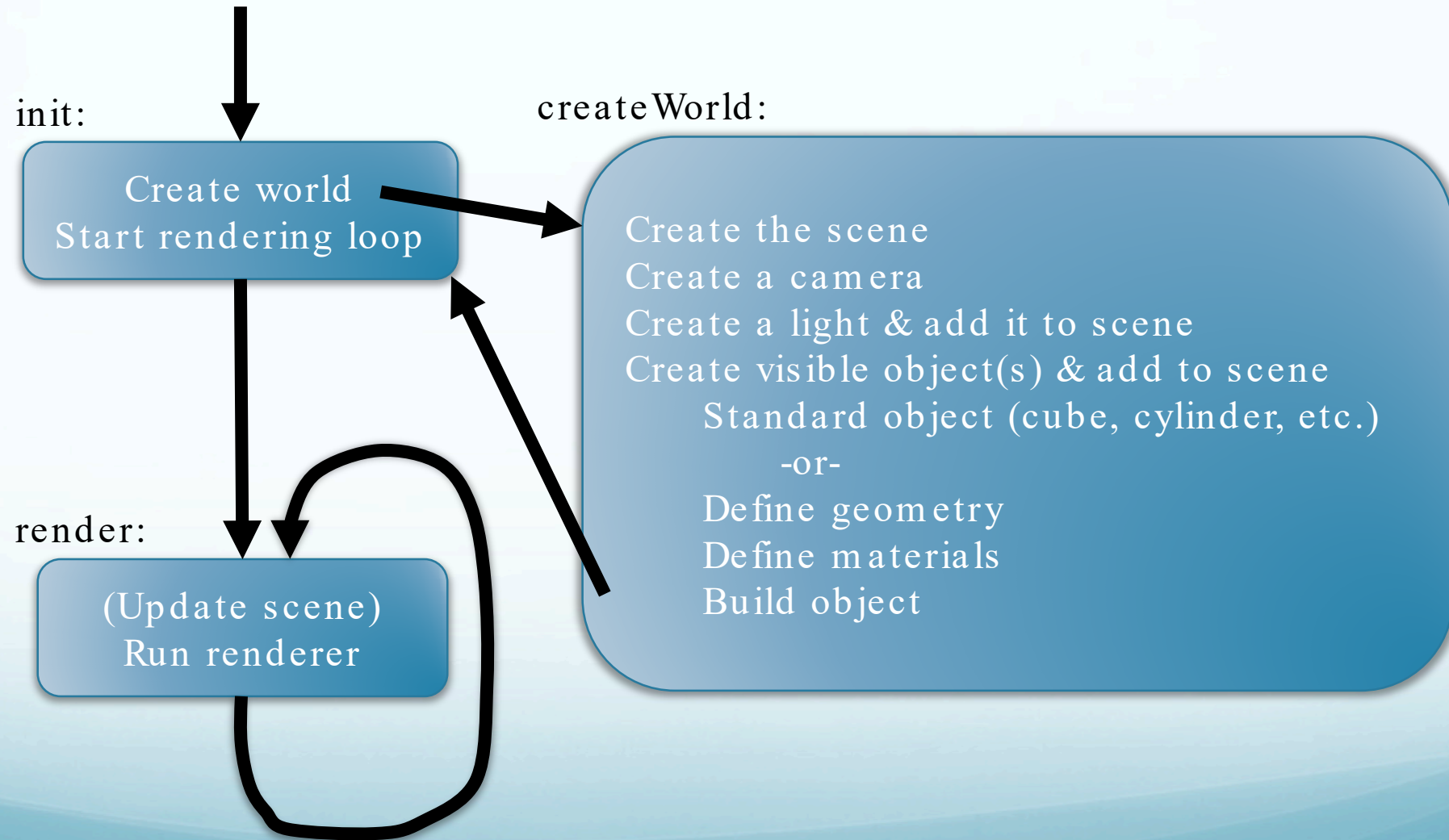
**Mesh** is used to create all sorts of solid objects

- Combination of **geometry** and **material**



```
// Create an object with prepared geometry and materials  
var myObject = new THREE.Mesh( myGeom, myMaterial );  
scene.add(myObject);
```

# Putting It Together In Code

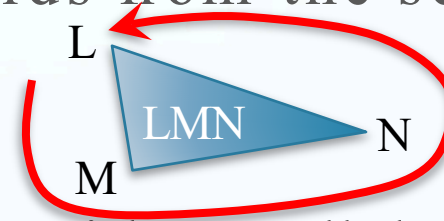
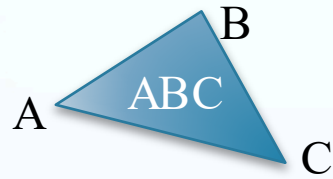




# Questions

PAUSE NOW & ANSWER

1. Which of the polygons below is facing outwards from the screen?



2. Which light best simulates sunlight on earth, with parallel rays?

*DirectionalLight*

3. Which of the following is NOT a predefined geometry offered by Three.js?

a. SphereGeometry

b. ~~CubeGeometry~~ *BoxGeometry*

c. ConeGeometry

d. CylinderGeometry

e. TorusGeometry

f. IcosahedronGeometry

# Review

After watching this video, you should be able to...

- Set up a web page for 3D rendering using Three.js
- Add a camera, lights, and visible objects to your scene
- Make informed choices about the different available options
- Create custom object geometries and combine them with materials
  - Write code to define vertices and combine them into faces
  - Control the directionality of triangular faces