Lecture 17
Meshes and Other Stuff

Last Time

- Rotations
- Meshes

Today

- Stuff we didn't get to last time
- How we deal with colors
- How this works in THREE

Meshes

In Computer Graphics...

A **Mesh** is a collection of (connected) triangles

In THREE...

A `Mesh` is a class that represents a geometric object

- It has a `BufferGeometry` - which is the collection of triangles
- It has other stuff it needs to be an Object3D
- It has a `Material`

BufferGeometry

Store a collection of triangles

- A list of vertices, each vertex has **attributes**
- Connectivity (how the vertices are connected)

Data stored in blocks of memory

- `BufferAttributes` - data with fixed layout
- each attached to some "name"

Whatever attributes the material will want/need

```
const geom = new T.BufferGeometry();

const mem = new Float32Array([/* 4 verts * 3 vals/vert = 12 numbers*/] );
const buf = new T.BufferAttribute(mem,3);
geom.setAttribute("position",buf);

const cmem = new Float32Array([ /* 12 numbers */]);
geom.setAttribute("color", new T.BufferAttribute(cmem,3));

const nmem = ... /** set up array of normals */;
geom.setAttribute("normal", new T.BufferAttribute(nmem,3));

// and so on...
```

Triangles from vertices

1. Triangle soup

   (v0,v1,v2), (v3, v4, v5), ...

2. Indexed

   `setIndex` - takes a list of vertex numbers (integers)

   technically its a buffer (3 verts/triangle, 1 integer per vertex)

A Bit of THREE History

1. `Geometry` - flexible, JavaScript data structures, easy to use
2. `BufferGeometry` - efficient, maps to how the hardware works

Two versions of everything

- `SphereGeometry` and `SphereBufferGeometry`

Need to convert `Geometry` to `BufferGeometry`

`Geometry` was deprecated

What does this mean for class?

I am trying to get rid of references to `Geometry`

- sometimes things sneak in

Back to Meshes...

How do we deal with colors?

1. Colors for the Object (in its `Material`)
2. Colors for each vertex (if the `Material` knows about them)
3. Texture colors (coming soon)

Note: Face (triangle) colors aren't on this list! (deprecated)

If you want to color a triangle you need to color its vertices

- vertex splitting

What colors do things appear?

Albedo - the color of the surface

- the color the surface reflects

Color of the object, color of the light - combine

- componentwise multiplication
- highlights change the color
  - or surface has separate specular color

Aside... Colors in THREE

Everything is `class Color`

Internally...

- it stores RGB

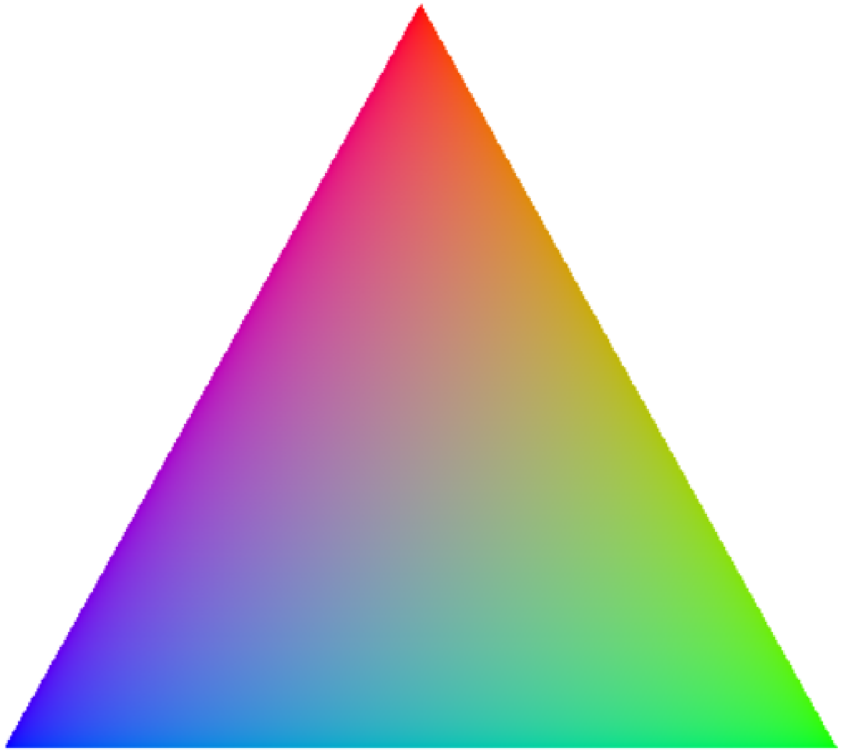
Externally

- get / set any way you like
- `.setRGB` (three numbers 0-1), `.setStyle` (CSS string)

Vertex Colors

```
let material =
   new T.MeshStandardMaterial({vertexColors:T.VertexColors});
```

Barycentric Color Interpolation



Barycentric Interpolation

Barycentric interpolation (over a triangle)

$$\mathbf{p} = \alpha\mathbf{A} + \beta\mathbf{B} + \gamma\mathbf{C}$$

where $\alpha + \beta + \gamma = 1$

Gives a **coordinate system**

- for the triangle ($\alpha, \beta, \gamma \in 0 - 1$)
- for the plane

Interpolating Colors (and other Vertex Properties)

Barycentric interpolation

$$\mathbf{p} = \alpha \mathbf{A}_{\mathbf{pos}} + \beta \mathbf{B}_{\mathbf{pos}} + \gamma \mathbf{C}_{\mathbf{pos}}$$

so...

$$\mathbf{color} = \alpha \mathbf{A}_{\mathbf{color}} + \beta \mathbf{B}_{\mathbf{color}} + \gamma \mathbf{C}_{\mathbf{color}}$$

More about Normals

Triangles (should) have an **outward** facing normal vector

We can compute this by the cross product

- **if** the vertices are ordered correctly

Why Specify Normals?

- specify outward direction if it isn't obvious
- fake normal directions (pretend a triangle is something else)

Outward Normals?

Assumes there is an inside and outside

- front and back of a triangle

By default, THREE only draws the front of a triangle

- need to tell the materials otherwise

Three's Compute Normals

- compute normals averages the triangles around the vertex

Uses of Normals

1. Backface Culling

   THREE.js does backface culling by default
   use `side: THREE.DoubleSide` with your materials for planes

   warning: doesn't use normals - uses triangle winding direction

2. Lighting

Transforming Normals

If we transform the **points of a triangle** what happens to its **normal**?

It is a **different** transformation!

- only the 3x3 matrix part (normals are vectors, translations don't matter)
- **adjoint** of the 3x3 part of the transform

The adjoint is the **inverse transpose**

For a rotation, the inverse transpose is the matrix itself

- this is only true for rotations!

Mesh Summary

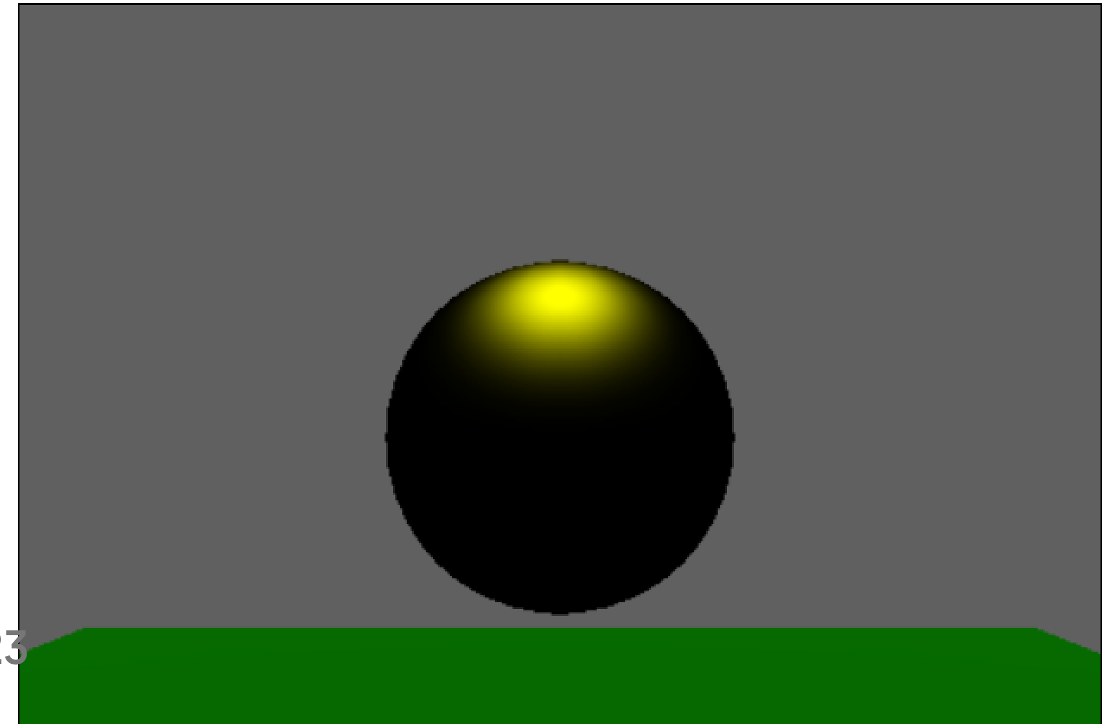
- Good Meshes
  - avoid cracks and T-Junctions
  - avoid bad triangles
  - consistent normals
- Data Structures for Efficient Sharing
- Vertex Properties / Vertex Splitting
- Basic Data Structures
- Buffers, AttributeBuffers and BufferGeometry
- Normals

Using Normals for Lighting

Since we skipped some details

Diffuse

Specular



Historic Models

Used in graphics for decades

First in the 1970s - balance efficiency and quality

Built into hardware in the 1980s

Standard in systems in the 1990s-2000s

Now with GPUs we can do better

Things still built on these models (understanding them is good)

Diffuse Lighting

Lambertian Materials

- scatters light in all directions
- doesn't matter what direction you look from

Diffuse Reflection

$$r_{\text{diffuse}} = \hat{\mathbf{n}} \cdot \hat{\mathbf{l}}$$

where:

- r_{diffuse} = amount of diffuse reflection
- $\hat{\mathbf{n}}$ - unit surface normal
- $\hat{\mathbf{l}}$ - unit vector to light source

using this...

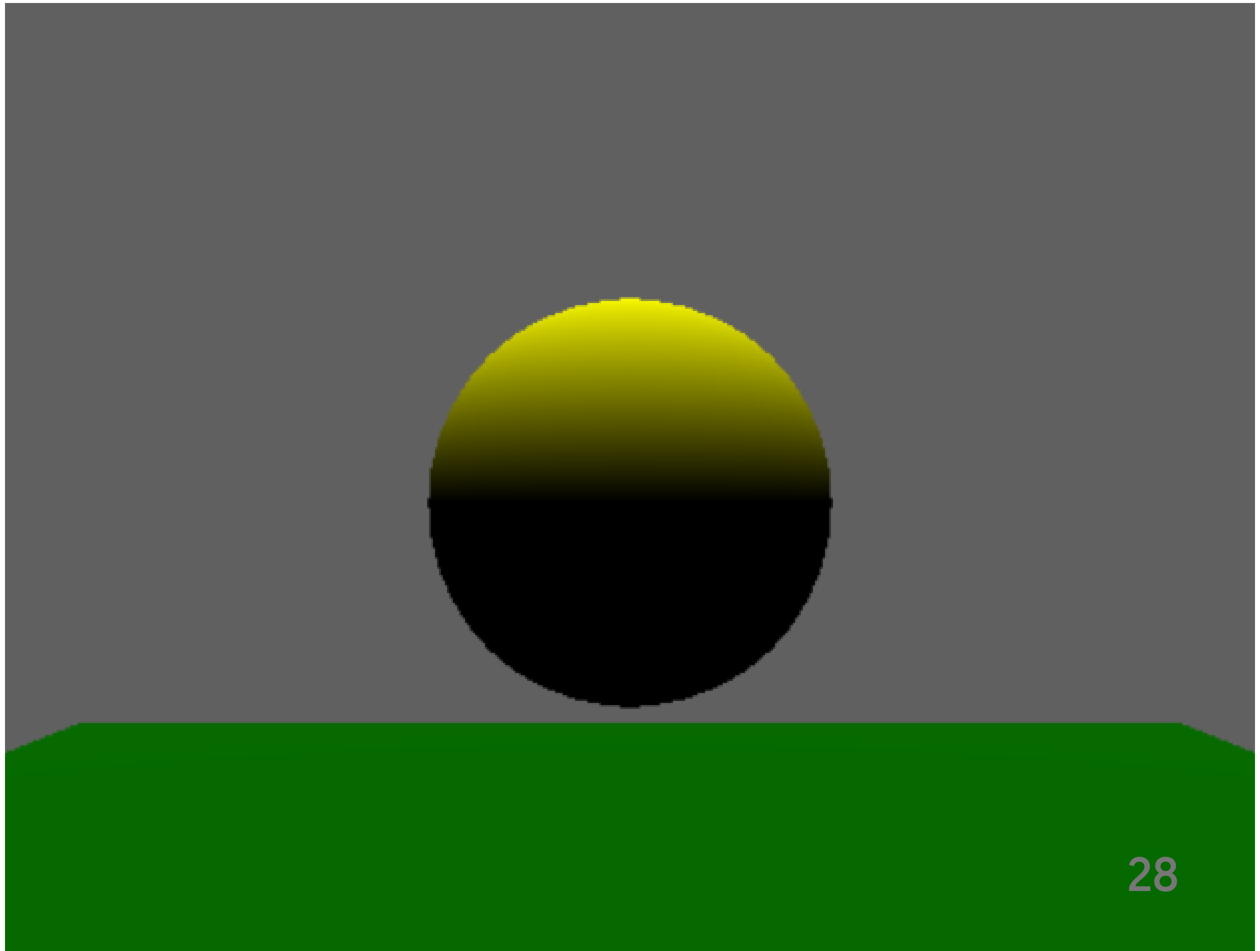
$$\mathbf{color} = (\hat{\mathbf{n}} \cdot \hat{\mathbf{l}}) \quad \mathbf{c}_{\mathbf{light}} \;\; \mathbf{c}_{\mathbf{d}}$$

where

- $\hat{\mathbf{n}}$ - unit surface normal
- $\hat{\mathbf{l}}$ - unit vector to light source
- $\mathbf{c}_{\mathbf{light}}$ - color/intensity of light
- $\mathbf{c}_{\mathbf{d}}$ - color of the material (diffuse reflectance)

looking at a sphere...

why a sphere is a good "test probe"



Shiny Things

We are reflecting the lights (for now)

Specular reflection

A Perfect Mirror

- Light direction and eye position matter
- The eye needs to be in the exact correct position
- $\hat{\mathbf{e}}$ (eye vector) and $\hat{\mathbf{r}}$ (reflection vector)

A Realistic (Imperfect) Mirror

Phong model - it's a hack!

Graduate fall off as we get away from the optimal direction

$\hat{\mathbf{e}} \cdot \hat{\mathbf{r}}$

keep > 0

Phong Model

Raise to the "power" of "shininess" (phong exponent)

$$r_{specular} = (\hat{\mathbf{e}} \cdot \hat{\mathbf{r}})^p$$

where

- $\hat{\mathbf{e}}$ is the eye vector
- $\hat{\mathbf{r}}$ is the reflection vector
- p is the "shininess" (material property), phone exponent
- $r_{specular}$ is the amount of specular reflection

Shinier = more like a perfect mirror

An Alternate Way to Compute

$$r_{specular} = (\hat{\mathbf{n}} \cdot \hat{\mathbf{h}})^p$$

where

- $\hat{\mathbf{n}}$ is the normal vector
- $\hat{\mathbf{h}}$ is the half-way vector (between $\hat{\mathbf{l}}$ and $\hat{\mathbf{e}}$)
- p is the "shininess" (material property), phone exponent
- $r_{specular}$ is the amount of specular reflection

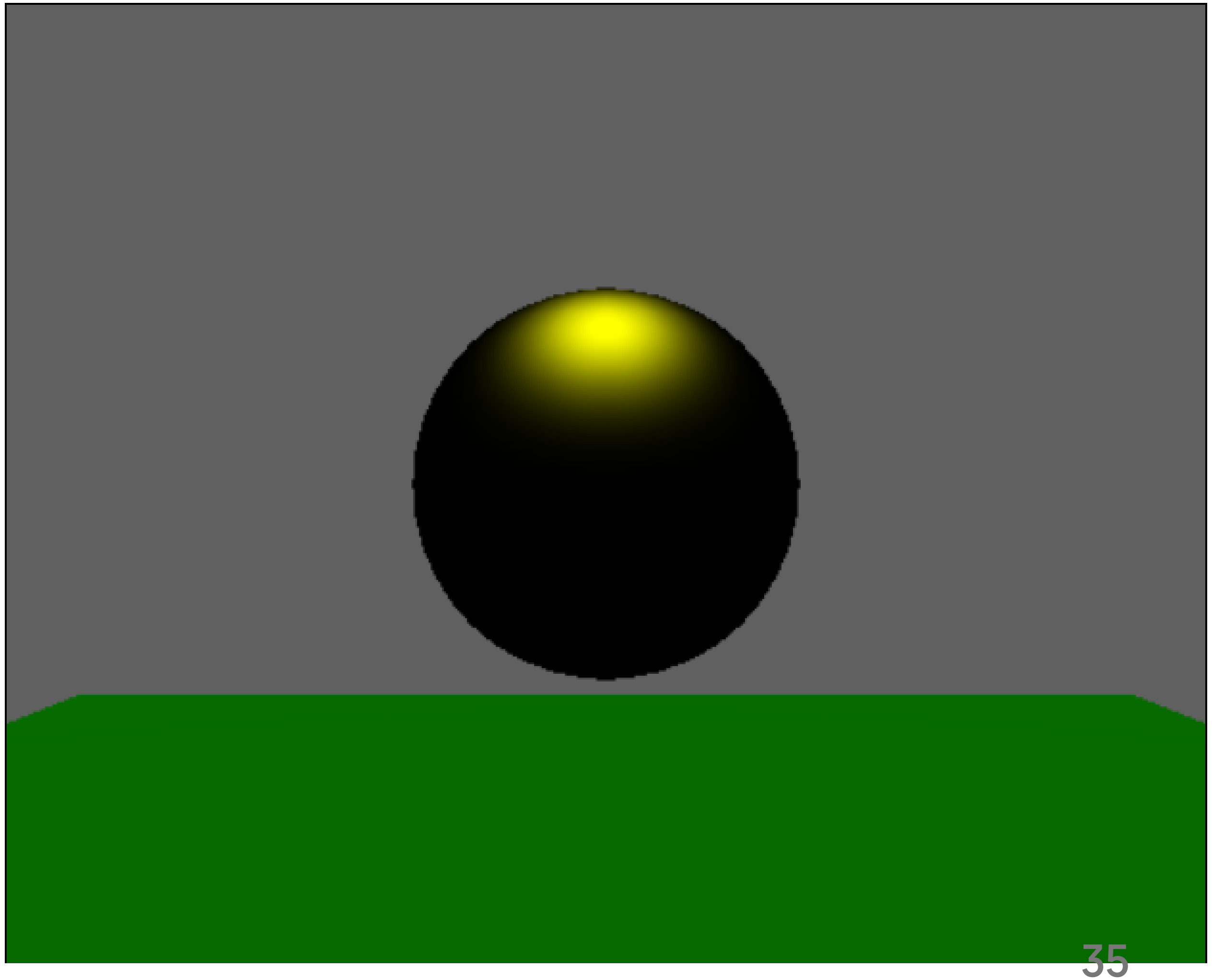
Using This

$$\mathbf{color} = (\hat{\mathbf{n}} \cdot \hat{\mathbf{h}})^p \quad \mathbf{c}_{\mathbf{light}} \quad \mathbf{c}_{\mathbf{s}}$$

where

- $\hat{\mathbf{n}}$ - unit surface normal
- $\hat{\mathbf{l}}$ - unit vector to light source
- $\mathbf{c}_{\mathbf{light}}$ - color/intensity of light
- $\mathbf{c}_{\mathbf{s}}$ - color of the material (specular reflectance)
- p - shininess of the material

Demo



How to make things look better

1. Better local lighting models (and materials)
2. Better lighting effects (global transport)
   - reflections (the world is a light source)
   - shadows
3. Better colors on the objects

Implementing #2 is hard... so we use hacks based on #3

Next up...

Textures

1. How to get more than 3 colors on a triangle
2. How this works
3. How to make this machinery do other things
    - fake surface complexity (non-smooth objects)
    - fake reflections
    - fake shadows

The Texture "Lectures"

(last year's videos)

Workbook 8 (Lect 17)

17.A - "Review" (really overview)

17.B - Texturing Basics

17.C - Texturing in THREE

17.D - How Texturing Works

Workbook 9 (Lect 18)

18.A - (none - became 17A)

18.B - Fake Normals (Bump Maps)

18.C - Other tricks

18.D - Environment maps

18.E - Shadow maps