UCLA

CS174A : Introduction to Computer Graphics

Kinsey 1240
MW 4-6pm

Scott Friedman, Ph.D
UCLA Institute for Digital Research and Education

Next Monday's Mid-Term

- Exam Option – 50 points possible
  - Multiple choice
  - Covers topics through basic texture mapping
  - Textbook, class notes are good study guides
  - Open book, notes
  - Closed computers and phones

Next Monday's Mid-Term

- "Hack-a-thon" Option – 200 points possible
  - You can work alone or with a partner
    - **MUST** send me message electing this option by Friday 2/10
      - Send to: `friedman@ucla.edu`
      - Include: Your UID(s) and GitHub username(s)
  - Four hours, 4pm - 8pm
    - Plan accordingly

Next Monday's Mid-Term

- "Hack-a-thon" Option
  - **Topic: Visualize data of your choice**
  - Not a lot of time
    - Think about what to do ahead of time
    - Line up API key(s), data sources
  - Static site – no server component
    - Pull resources from the web
    - Use static data
  - Keep it Simple, Be Creative, Be Fun

Next Monday's Mid-Term

- "Hack-a-thon" Option
  - Point availability:
    - 100 points for incorporating class topics
      - Through lighting and texture mapping
      - Code quality and comments (a mess will not do)
    - 50 points from me
      - Overall quality of (partial) solution
    - 50 points from other hack-a-thon participants
      - You will have 48 hours to submit
      - Overall quality of (partial) solution
  - We will keep the 1 and 2 people teams as separate categories

Next Monday's Mid-Term

- Do Nothing Option – 0 points
  - Take the day off

Assignment #3

- Texture Mapping
  - Very simple exercise applying texture maps
  - Posted tonight
  - Due 2/25

Animation

- Want to continuously animate some geometry
- One way
  - Update position by some (small) amount
  - Call requestAnimFrame() at the end of your rendering code.
  - This sort-of works

- There are two problems with this
  - You cannot assume the rate at which the browser will respond
    - It may or may not be 60 frames per second (fps)
  - You may be on a slow or heavily loaded machine
    - So even if the browser would normally update at 60fps it doesn't under circumstances such as these.

Animation

- Only under ideal circumstances will you
  - Be able to update your movement x/60
  - And get a matching visual result in time
  - Your movement will be slower or faster than you expect – or variable!
  - Time in your application and real time will diverge.
    - Its distracting
- A better way
  - Use a timer! setTimer() or setInterval()
  - Even the requestAnimFrame wrapper falls back to this when all else fails.
    - Right?

Animation

- A better way, cont…
  - Problem with setTimer is that it's a little too heavy handed
    - It will always fire – even when the window/tab is not visible.
    - It is not synchronized to the display refresh
    - It can also have some variability with respect to real time
  - It's a fall back when there is no choice (which is rare)

- A better way, part 2
  - requestAnimationFrame() calls your callback with a parameter!
  - That parameter is the (hires) time when the callback is invoked.
  - Now you can update your position based on absolute time and get smooth animation regardless of the browser frame rate

Animation

- Things to consider
  - You probably want to have an initial time to work with in order to compute a delta time
    - Let’s say you want to rotate a 0.25 rotations per second
    - For each frame you would update 0.25 x dT (assuming dT is in seconds)
    - Setup

```
var dT = null
function render( timestamp )
{
    if ( !dT ) dT = timestamp;
    dT = timestamp - dT;
    // draw your frame
    window.requestAnimationFrame( render );
}

window.requestAnimationFrame( render );
```

  - Once you call requestAnimationFrame(), *it will call your callback*
    - Unless you cancel it using cancelAnimationFrame()
    - requestAnimationFrame() returns an ID you can use to cancel.
    - Each invocation queues a callback – if you call it five times, you will get your callback invoked five times.

Animation

- Things to consider
  - What's the difference between
    `window.requestAnimationFrame()`
    `window.requestAnimFrame()`
  - Nothing, the second is in the webgl-utils.js file and handles variation in the function's invocation between browsers.
    - This may affect the precision of the timestamp parameter – see link below
  - Not actually part of WebGL
    - Part of the functionality of the `window`
  - There are some good links to check out at the bottom of this page
    - https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame

More Lighting

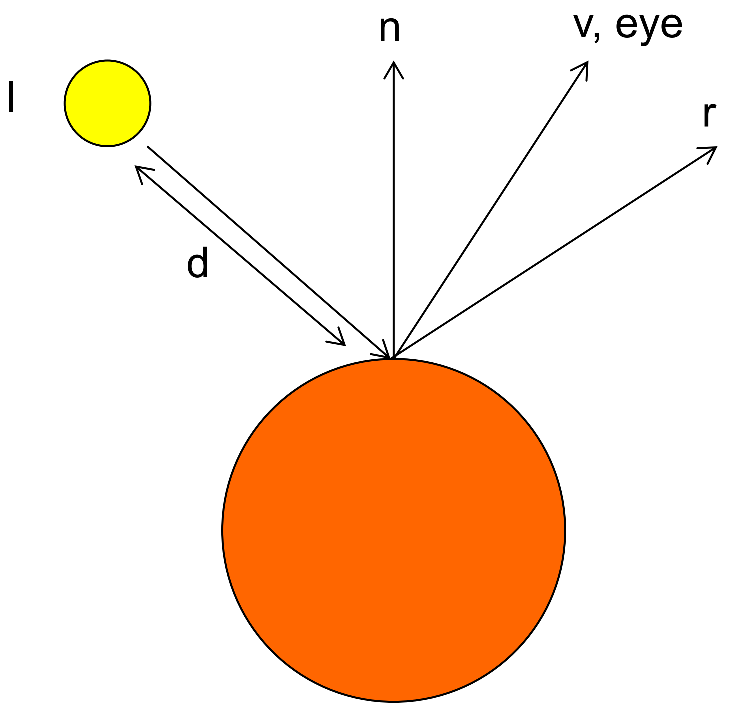
- The **Phong lighting/reflection** model and **Phong shading** are *two different things*

- **Phong Lighting/Reflection Model**
  - How we model the different aspects of light interaction with a surface in computer graphics, e.g Ambient, Diffuse and Specular

- **Phong Shading (Phong/Blinn)**
  - A method of determining the normal of a primitive by interpolating between the vertex normals over the surface of the primitive.

More Lighting

- *All* of the techniques we talked about use the Phong reflection model
  - Notice that **I** needs to be reversed when performing the actual lighting calculation.



More Lighting

- Make sure all vectors are normalized
- Make sure all vectors are pointing in the right direction
  - Otherwise, weird things will happen with you lighting!
- Simplify, leave out quadratic, reflection coefficients, *k*, to start to help get things working.

$$I = \frac{1}{a + bd + cd^2}(k_d L_d \max(\mathbf{l} \bullet \mathbf{n}, 0) + k_s L_s \max((\mathbf{r} \bullet \mathbf{v})^\alpha, 0)) + k_a L_a$$

- Try using 4.0 for the specular power term
- Review chapter 6 in textbook – what you need is there

More Lighting

- Where are the colors and normals?
  - How many are needed?
  - Where do they go in my code?
- Where to perform lighting calculations?
  - In Javascript or the GPU/shaders?
  - Want to or have to…
  - Efficiency

More Lighting

- Where are the colors and normals?

- Colors come from a material and light definition
  - A data structure you define
  - You do not necessarily need a color buffer (attribute) to go with vertices
  - You do need normals though
  - Getting the normals *right* is the challenge
- There is only a single light in assignment 2
- Multiple materials
  - But only one will be applied at a time
  - Use uniforms to pass into shaders

More Lighting

- Where are the colors and normals?
- Normals are needed for all *lit* geometry
  - Do we need normals for all vertices?
    - …in a buffer? How about flat shading?
  - Is the sun lit?
    - Does it need normals?
  - Could you need more than one normal per vertex?
    - Would that mean repeating some vertices?
    - Could save space using element arrays (indexed arrays)
- Normals do need a buffer

More Lighting

- Where to perform lighting calculations?
  - It depends

- Three things to consider
  - What do you need to know and when do you know it (for lighting)
  - What type of shading are you performing?
  - Efficiency of computation

More Lighting

- Flat, Gouraund and Phong Shading
  - How many normals do you really need?
    - Per primitive? Per vertex?
    - Theoretical or implementation point of view?
      - Flat shading only needs one normal per primitive, in theory
      - How does the implementation work?
  - When and where can that normal be computed?
    - Javascript
    - Vertex shader (hint)
  - When and where can the lighting be computed?
    - Does it always need to be in the fragment shader?

More Lighting

- Flat Lighting
  - One normal per primitive (triangle)
    - How are they computed
    - Could you have more than one? Per vertex?
  - One color per *primitive* computed from lighting
  - Lighting computed in Javascript or GPU (shader)
    - GPU – in vertex or fragment shader
    - Which is most efficient?
    - If neither the light or object ever moved you could compute the lighting once!

More Lighting

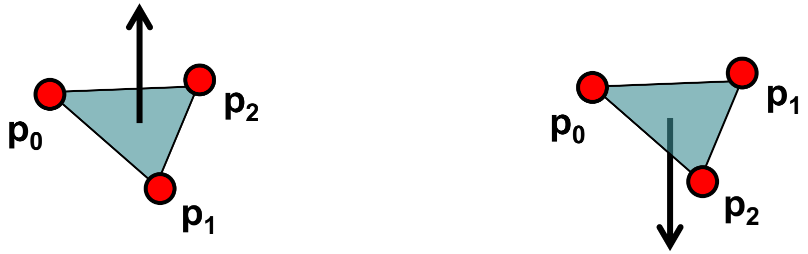
- Gouraund Lighting
  - Three normals per primitive (triangle)
    - How are they computed?
  - One color per *fragment* computed from lighting
  - Lighting computed in Javascript
    - Not really, why?
  - Lighting computed in GPU
    - Vertex shader?
      - Yes, why?
    - Fragment shader?
      - No, why?

More Lighting

- Phong Lighting
  - Three normals per primitive (triangle)
    - How are they computed?
  - One color per *fragment* computed from lighting
  - Lighting computed in Javascript
    - Not really, why?
  - Lighting computed in GPU
    - Vertex shader?
      - No, why?
    - Fragment shader?
      - Yes, why?

Computing Vectors

- Shading – The Normal
  - For flat surfaces we mean the normal to a plane.
  - This is a vector we have seen already.
  - For three non-collinear points, the normal is
  - The order we consider the points *matters.*
  - It will *determine* the direction of the normal **n**.
  - Doing it backwards may cause you to not see anything!



Computing Vectors

- Make sure you get the vertex order correct!

- Why?
  - because vectors are directional
  - The order of the points used to create them affects the direction
  - Subtracting two points, v - p is not the same as p - v

Caution!

- There are tons of examples out on the internet
- Be careful when lifting code to make sure you understand what it is doing
- Trying to untangle someone else's code that you modified to do something you don't understand is a recipie for
  - Long nights
  - Tears
  - A blank screen
- Ask questions, start simple and build up.