Lighting, Part II

Motivation

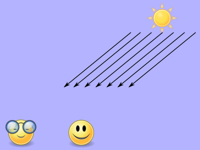
- See some additional lighting constructs
- Lighting is important for realism, so we want to pay close attention to it

Review

- Basic lighting computations
- Ambient, Diffuse, Specular

Directional

- Some lights are (or can be treated as being) infinitely far away
  - Ex: Sunlight
- This changes our lighting computations
- Right now, we have: vec3 L = normalize(lightPosition - v_worldPosition)
- But if the light is infinitely far away, L is the same for all object points
- Convention: We treat light's "position" as the direction *to* the light in that case



Uniforms

- We could define some uniforms:
  - vec3 lightPosition;
  - float positional; //0 or 1

Code

- And then the computation:

```
void main(){
    ...
    vec3 L;
    if( positional == 1.0 )
        L = normalize(lightPosition - v_worldPosition);
    else
        L = lightPosition;
    ...
}
```

Problem

- One problem: if-tests can be a little expensive
  - System might execute *both sides* of the if-else and then discard one of the results
  - It’s not too bad in this case...
  - But we can do better!

Code

- Code it like this instead:
  ```
  vec3 L = normalize(lightPosition - positional * v_worldPosition);
  ```
- This is a common pattern in GPU programming
  - We can often reduce an if-else test to a multiplication, if we choose our values wisely

In Fact...

- We can tidy things up a bit
- Suppose we define lightPosition as a *vec4*
  - If w==1: Positional light, xyz=location
  - If w==0: Directional light, xyz=direction to the light
- Then we compute:

```
vec3 L = normalize( lightPosition.xyz -
                    v_worldPosition*lightPosition.w);
```

Attenuation

- Right now, our light will shine an infinite distance
- That's not realistic!
  - Can you stand on the ground and point a flashlight at the moon and see the spot?
- Light *attenuates* with increasing distance
  - Attenuate = Less of it
- Note: Attenuation only makes sense with positional lights (not directional ones)

Attenuation

- Let d be distance from light to surface point
  - We can get this easily: float d = distance(lightPosition, v_worldPosition);
- Typically, we have three parameters: A_0, A_1, and A_2
  - These could be uniforms or we could #define constants
  - A_0 tells constant falloff; A_1 tells linear falloff; A_2 tells quadratic falloff
- Compute intensity as usual...
  diffusePct = ... ;
  specularPct = ... ;

Attenuation

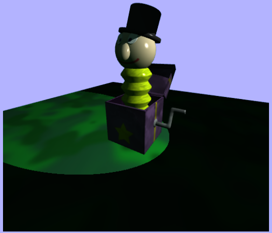
- Let $f = \frac{1}{A_2 \cdot d^2 + A_1 \cdot d + A_0} = \frac{1}{d(d \cdot A_2 + A_1) + A_0}$
  - Note the use of Horner's rule to save a multiply
- Ensure f is at least 1.0:
  f = min(1.0,f);
- Scale diffuse and specular lighting:
  diffusePct *= f;
  specularPct *= f;

Example

- Attenuation explorer: https://www.ssucet.org/~jhudson/19/2801/attenuationExplorer/

Spotlight

- We might want to simulate a spotlight
- Note: Spotlight $\neq$ Directional light!
  - Directional lights *cannot* be spotlights

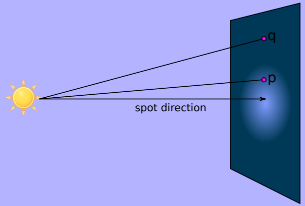


Spotlight

- Examples:
  - Theater (stage) lights
  - Vehicle headlights
  - Searchlights
  - Flashlights
- Light doesn't radiate in all directions

Spotlight

- Need two pieces of information (uniforms):
  - Spotlight direction: Central axis of light (vec3)
  - Spotlight opening half-angle (scalar)
- If angle between spot direction and vector from light to surface point > spot angle: Not lit
- Else: Lit



Spotlight

- Could compute angle with dot product and acos:
  L = normalize(lightPosition - v_worldPos);   //FROM surface TO light
  float angle = acos(dot(spotDirection, -L));
- But acos can be slow!

Spotlight

- Better: Pass in *cosine* of spot opening angle as a uniform:
  ```
  float cosineMaxSpotAngle;
  vec3 spotDirection;
  ```
- Then compute like so:
  ```
  float spotDot = dot(-L,spotDirection)
  if( spotDot >= cosineMaxSpotAngle )
      lit
  else
      unlit
  ```

Note

- What if we want to support both spotlights and non-spotlights?
- We *could* have uniform for “is/is not a spotlight”
- And then do if-else test
- But is there another way so we don’t need an if-else test for this?

Spotlight

- Solution: If light is omnidirectional, set cos_spot_angle to anything <= -1
- Then dot product in the if-test will always say “lit”
- And so we’ll never have a spotlight cutoff

But...

- We can go further and trim the single if-test that we have as well!
- Let spotDot = dot(-L,spotDirection)
  - Gets smaller as we move away from where light is shining
- bool spotB = (spotDot >= cosineMaxSpotAngle);
  - Once we move outside light cone, spotB becomes false
- Convert: float spotF = float(spotB)
  - spotF becomes 0 (if spotB is false) or 1 (if spotB is true)
- Now multiply spotF by diffusePct and specularPct
  - If we're outside the illumination cone, we get no illumination

Fadeout

- Maybe we don't want an abrupt spotlight boundary
  - Make spotlight fade out as we approach the edge
- Define a constant cosineSpotFadeAngle > cosineMaxSpotAngle
  - If spotDot > cosineSpotFadeAngle set spotF=1
    - We're inside the fully lit region
  - If cosineSpotFadeAngle > spotDot > cosineMaxSpotAngle: spotF goes smoothly from 1 to 0
    - We're in the fadeout region
  - If cosineMaxSpotAngle > spotDot: spotF is 0
    - Outside spotlight

Fadeout

- spotDot = dot(-L,spotDirection)
- spotF = clamp((spotDot - cosineMaxSpotAngle) / (cosineSpotFadeAngle - cosineMaxSpotAngle), 0.0, 1.0)

Example

- Spotlight explorer: `https://www.ssucet.org/~jhudson/19/2801/spotlightExplorer/`

Several Lights

- What if we want several lights?
- We could create a bunch of uniforms:
  - vec3 lightPosition0;
  - vec3 lightPosition1;
  - vec3 lightPosition2;
  - etc.
- This is tedious!

Better Way

- Better way: Create arrays for the data
- Pitfall: GL pads each array element out to 4 floats
  - Essentially, every array is treated as being array of vec4's, even if we don't use four components
  - Wastes some memory
  - Not much RAM available for uniforms: GPU's might limit to only 256 vec4's!
  - We're already using some uniforms for matrices (4 for worldMatrix, 4 for viewMatrix, 4 for projMatrix), eyePos, etc.
- So we are motivated to *pack* our data

Arrays

- Define:
  - lightPositions: Array of vec4: xyz=position (or direction); w=1 for positional or 0 for directional
  - lightColors: Array of vec4: xyz=rgb; w=cosine of spot fade angle
  - spotDirections: Array of vec4: xyz = spot direction, w=cosine of max spotlight angle
- Concept: Packing several unrelated items into one variable
  - Commonly seen in GPU programming

Uniforms

- In uniforms.txt:

```
#define MAX_LIGHTS 8
vec4 lightPositions[MAX_LIGHTS]; //xyz=location, w=positional
    flag
vec4 lightColors[MAX_LIGHTS]; //color (xyz=color, w=cosine spot
    fade angle)
vec4 spotlightDirections[MAX_LIGHTS]; //xyz=spot direction, w=
    cosine spot angle
```

FS

```
vec3 N = normalize(v_normal);
vec3 totalDiffuseColor=vec3(0.0);
vec3 totalSpecularColor=vec3(0.0);
for(int i=0;i<MAX_LIGHTS;++i){
    float positionalOrDirectional = lightPositions[i].w;
    vec3 lightPos = lightPositions[i].xyz;
    vec3 spotDir = spotlightDirections[i].xyz;
    float cosineMaxSpotAngle = spotlightDirections[i].w;
    vec3 lightColor = lightColors[i].xyz;
    float cosineSpotFadeAngle = lightColors[i].w;
    vec3 L = normalize( lightPos - positionalOrDirectional * v_worldPos);
    float diffusePct = max( 0.0, dot(L,N) );
    float spotDot = dot(-L, spotDir );
    float spotF = clamp( (spotDot - cosineMaxSpotAngle) / (cosineSpotFadeAngle - cosineMaxSpotAngle), 0.0, 1.0
          )
    diffusePct *= spotF;
    totalDiffuseColor += diffusePct * lightColor;
    ...do specular too...
}
vec4 texColor = texture( tex, vec3(v_texCoord,0.0) );
color.rgb = ambient * texColor.rgb +
          totalDiffuseColor * texColor.rgb  +
          totalSpecularColor ;  //optional: multiply specular by texColor
```

CPU

- On the CPU side we have a bit more work to do
- Must teach Program.py how to handle uniform arrays
- And we must set the uniforms themselves
- New Program.py

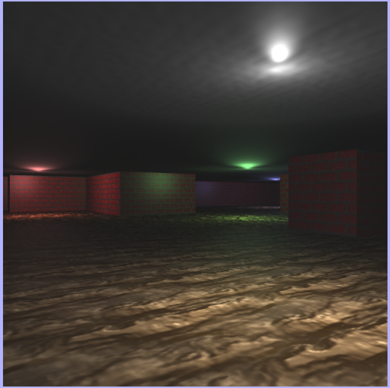
Mangement

- How to handle lights?
- Create a class to manage them:

```
class LightManager:
    MAX_LIGHTS=8        #must match what's #define'd in shader
    def __init__(self):
        self.positions  = [vec4(0) for i in range(MAX_LIGHTS)]
        self.colors     = [vec4(0) for i in range(MAX_LIGHTS)]
        self.spotlights = [vec4(0) for i in range(MAX_LIGHTS)]
    def setUniforms(self):
        Program.setUniform("lightPositions", self.positions)
        Program.setUniform("lightColors", self.colors)
        Program.setUniform("spotlightDirections", self.spotlights)
        Program.setUniform("attenuation", ... )
        Program.updateUniforms()    #if desired; more efficient to defer this if possible
    def setPosition( self, index, location, positional):
        vec4 tmp
        tmp.x = pos.x; tmp.y = pos.y; tmp.z = pos.z;
        if positional: tmp.w = 1
        else: tmp.w = 0
        self.positions[index] = tmp
    def setColor( self, index, color ):
        ...etc...
```

Assignment

- Add attenuation and four light sources, each of which has a different color



Bibliography

- D. Brackeen. Programming Games in Java.
- D. Hearn & M. P. Baker Computer Graphics in C.
- F. Luna. Introduction to 3D Game Programming with DirectX 9.0.
- F. Luna. Introduction to 3D Game Programming with DirectX 10.
- F. Luna. 3D Game Programming with DirectX 12. Mercury Learning.
- Eric Lengyel. Mathematics for 3D Game Programming & Computer Graphics. Charles River Media.
- https://en.wikipedia.org/wiki/Blinn%E2%80%93Phong_shading_model
- https://gaim.umbc.edu/2010/09/07/approximation/
- https://github.com/KhronosGroup/glTF/blob/master/specification/2.0/README.md#appendix-b-brdf-implementation
- Naty Hofman. Background: Physics and Math of Shading. http://blog.selfshadow.com/publications/s2014-shading-course/hoffman/s2014_pbs_physics_math_slides.pdf

Created using LaTeX.

Main font: Gentium Book Basic, by Victor Gaultney. See http://software.sil.org/gentium/
Monospace font: Source Code Pro, by Paul D. Hunt. See https://fonts.google.com/specimen/Source+Code+Pro and http://sourceforge.net/adobe
Icons by Ulisse Perusin, Steven Garrity, Lapo Calamandrei, Ryan Collier, Rodney Dawes, Andreas Nilsson, Tuomas Kuosmanen, Garrett LeSage, and Jakub Steiner. See http://tango-project.org