# Lab 4: Intro to Lighting & Shading Techniques

Publish Edit ⋮

**GPR-200: Introduction to Modern Graphics Programming**
**Instructor: Daniel S. Buckstein**
**Lab 4: Intro to Lighting & Shading Techniques**

**Summary:**
This lab introduces classic and efficient lighting algorithms, used to speed up lighting in real-time. We explore the pre-cursors to and simplified versions of modern ray-tracing methods.

**Submission:**
Start your work immediately by ensuring your coursework repository is set up, and public. Create a new main branch for this assignment. ***Please work in pairs (see team sign-up) and submit the following once as a team***:

1. Names of contributors
   e.g. **Dan Buckstein**
2. A link to your public repository online
   e.g. **https://github.com/dbucksteinccorg/graphics2-coursework.git**
   (note: this not a real link)
3. The name of the branch that will hold the completed assignment
   e.g. **lab0-main**
4. A link to your video (see below) that can be viewed in a web browser.
   e.g. <insert link to video on YouTube, Google Drive, etc.>

Finally, please submit a ***5-minute max*** demo video of your project. Use the screen and audio capture software of your choice, e.g. Google Meet, to capture a demo of your project as if it were in-class. This should include at least the following, in enough detail to give a thorough idea of what you have created (hint: this is something you could potentially send to an employer so don't minimize it and show off your professionalism):

- Show the final result of the project and any features implemented, with a voice over explaining what the user is doing.

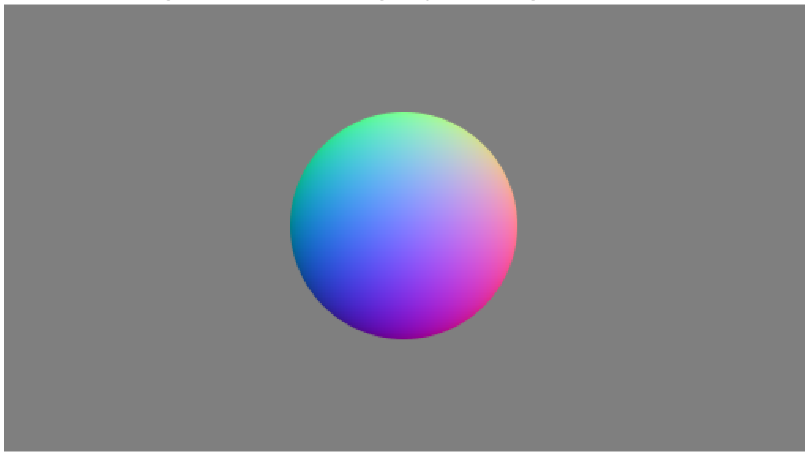
- Show and explain any relevant contributions implemented in code and explain their purpose in the context of the assignment and course.
- Show and explain any systems source code implemented, i.e. in framework or application, and explain the purpose of the systems; this includes changes to existing source.
- **DO NOT AIM FOR PERFECTION, JUST GET THE POINT ACROSS**. Please mind the assignment rubric to make sure you have demonstrated enough to cover each category.
- **Please submit a link to a video visible in a web browser, e.g. YouTube or Google Drive.**

**Objectives:**
The outcome of this assignment is a simple scene with simulated objects lit with fundamental lighting algorithms. You will learn a variety of ubiquitous lighting vocabulary, algorithms and techniques.

**Instructions & Requirements:**
***DO NOT begin programming until you have read through the complete instructions, bonus opportunities and standards, start to finish. Take notes and identify questions during this time. The only exception to this is whatever we do in class.***
Using Shadertoy, implement a variety of efficient lighting methods for a scene of simulated 3D objects (e.g. spheres). You must demonstrate all of the following:

1. ***Scene objects***: Draw a bunch of colored spheres using some sphere rendering function or algorithm.
    - Copy the contents of the Lab 4 starter text file into a new Shadertoy to help you get started.
    - *Ray-tracing*: Use the ray-tracing process to display spheres (see up to chapter 6 of the ray-tracing book). You should have spheres with surface normals displayed.
        - The geometric surface of the sphere is calculated using the ray position evaluation function (measured in arbitrary scene *units* because we have established the size of the viewing plane in the scene):
          *position_frag = (origin_ray) + t*(direction_ray);*
            - ***How can we optimize this for repeated use? Hint: don't reimplement the formula every time you want to use it...***
        - The geometric normal of the surface at this position on the sphere is calculated as the normalized difference from the center of the circle to the point:
          *normal_frag = normalize(position_frag - center_sphere);*
            - In this case, the center of the sphere is the same pixel coordinate of the circle on-screen.
            - ***What is the most efficient way to normalize this particular vector? Hint: it's not the 'normalize' function...***
    - *Procedural:* In image space, you can determine if a point lies on an imaginary sphere by using Pythagorean theorem.
        - First, determine if the current pixel lies within a circle on the screen. The test for this is: if the distance from the circle center to the current point is less than the radius, then the point is contained in the circle:

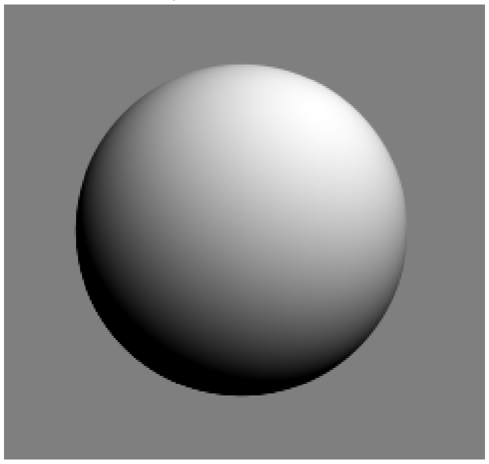
*dp.xy = coord.xy - center_circle.xy;*
*if (length(dp.xy) <= radius) inCircle = true;*
    - ***What is a more efficient way to test this?  Hint: 'length' is inefficient and requires prior calculation of some other metric using 2D Pythagorean theorem...***
  - Project the circle into 3D (a sphere) by calculating the Z component.  This is also done using Pythagorean theorem, by rearranging the terms:
    *r_sq = dx_sq + dy_sq + dz_sq*
    *dz_sq = r_sq - (dx_sq + dy_sq)*
    - Components beginning with 'd' represent a component of the difference seen in the comparison above; since that is 2D, the purpose of this is to calculate the missing Z component.
    - ***How can this be optimized further?  Hint: given the previous hint, you have already calculated a bulk of the components above...***
  - Take the square root of Z-squared to get the third component of the relative position vector (measured in same unit as whatever *coord* is used; e.g. *fragCoord* is in pixels, *viewport* coord is in arbitrary scene units):
    *position_frag.xyz = (center_circle) + vec3(dx, dy, sqrt(dz_sq));*
    - The *offset* is just some reference point in whatever space you're using.
  - The geometric normal of the surface at this position is calculated in the same fashion as with ray-tracing.
- Regardless of whether you generate geometry procedurally or using ray-tracing, the resulting scene should consist of one or more spheres, shown here with the normal for color against a solid grey background:



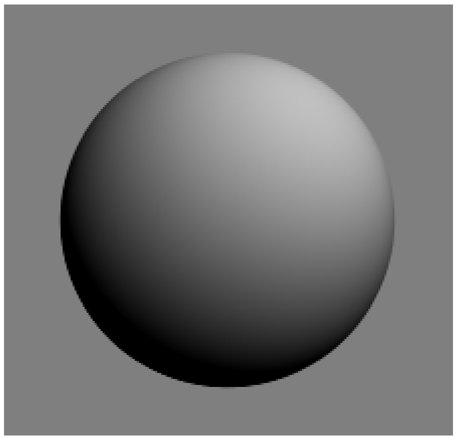
2. ***Setup lights***: We will explore lighting techniques using *point lights*, which have a consistent, spherical influence over space; they are represented as a center point and an intensity value.
   - Implement a data structure called *point light* or something of the like with at least the following data:
     - *Center:* the location of the light source in the scene.
     - *Color:* the color of the light source.

        - *Intensity:* a float describing the strength of the light (ultimately influences how far it will reach).
    - Instantiate a light as part of your scene.  You will use it to light the spheres.
        - Remember that the light's position and intensity must be measured in the same spatial unit as the spheres; if scene objects go by pixel coordinates, so do your lights.
3. ***Lambertian reflectance***: The Lambertian reflectance model efficiently calculates the *diffuse shading* at a point on a surface.  As described in the ray-tracing book (c. 8), diffuse shading is the result of rays (simulating a stream of photons) randomly bouncing off of surfaces that modulate the final color.  The intensity of the light accumulated is based on Lambert's cosine law, which, in summary, is that the intensity is a factor of the cosine of the angle of incidence between the ray and the surface.  The simulated Lambertian reflectance model in real-time is the simplification and approximation of this process; light is most intense when directly perpendicular to the surface, and has no effect when parallel to the surface.
    - ***Diffuse intensity***: Here we calculate the light's intensity at some point on the surface using one small formula (instead of a bunch of ray tests and bounces), specifically relating to the surface being drawn: $I_D = k_D I'_L$
        - $I_D$ is the diffuse intensity, the result of Lambertian reflectance;
            - $k_D$ is the diffuse coefficient, calculated using a simplified cosine law (see *diffuse coefficient*);
            - $I'_L$ is the attenuated intensity of the light at the surface (see *attenuation*).
    - ***Diffuse coefficient***: The diffuse coefficient is calculated as follows, using the geometric properties of cosine, each line explained below:
        - We begin with the a simplification of Lambert's cosine law: the diffuse coefficient is the cosine of the angle of reflectance, describing the overall collection of the light at the surface: $k_D = \cos(\theta_{\hat{N},\hat{L}})$
        - We already have the unit *surface normal* $\hat{N}$, but the *light vector* is calculated as the difference *from* the surface position *towards* the light's position, then normalized: $\hat{L} = \text{normalize}(P_{light} - P_{surface})$
            - The cosine of the angle between any two vectors (e.g. *a* and *b*) is related to *dot product* of the two inputs: $a \cdot b = |a||b| \cos(\theta_{a,b})$
            - If the inputs are unit-length (pre-normalized), then the cosine *is* the dot product: $\hat{a} \cdot \hat{b} = \cos(\theta_{\hat{a},\hat{b}})$
        - Therefore, substituting the unit surface normal $\hat{N}$ and unit light vector $\hat{L}$ yields this calculation for the diffuse coefficient: $k_D = \hat{N} \cdot \hat{L}$
            - Note that the range of cosine is both negative and positive, but the coefficient excludes negative values (which would imply the inside of the surface is lit); therefore we have a clamped coefficient: $k_D = \max(0, \hat{N} \cdot \hat{L})$

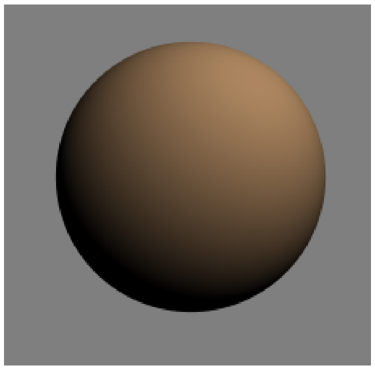
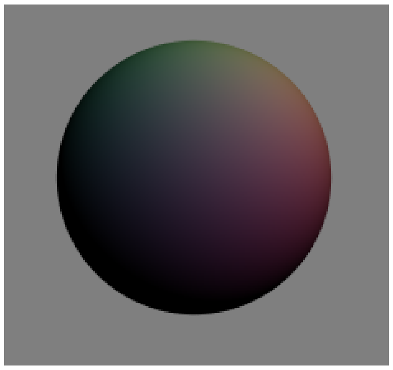
    - The result is a unit value, between 0 and 1, that is highest when the light source is perpendicular to the surface.
  - Here is the diffuse coefficient assigned to all color channels, given a light positioned just above and to the right of the viewer:



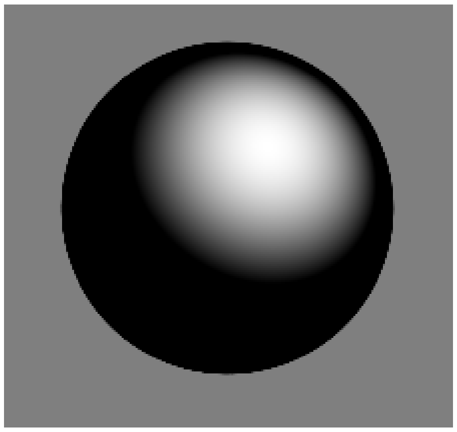
- ***Attenuation***: The light's intensity at the point on the surface is calculated using *attenuation*. Just as light fades with distance in real life, sometimes subtly, we can simulate this as well. There are many ways to do this (see if you can find more methods), but one very common attenuation function is:
$I_L' = 1/(1 + d/I_L + d^2/I_L^2)$
  - $I_L'$ is the attenuated intensity of the light;
    - $d$ is the distance between the light center and the surface point;
    - $I_L$ is the original intensity of the light (from data structure).
  - Here is the same scene as above with attenuation:

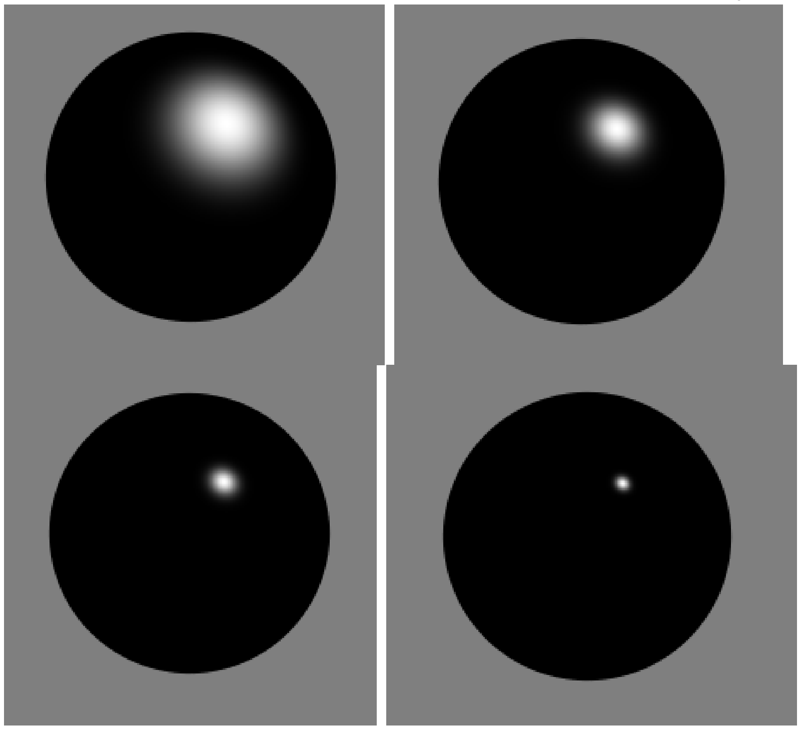


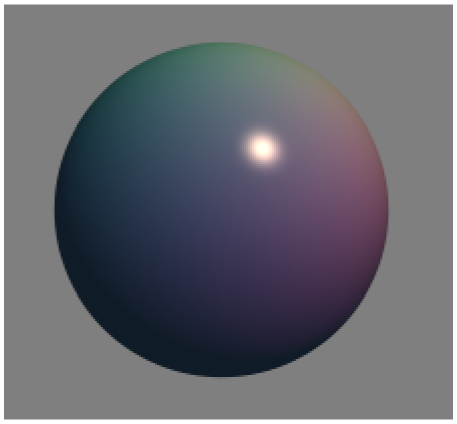
- ***Final color***: The final Lambertian color of the surface at the point being evaluated is calculated as: $C = I_D C_D C_L$
  - $C$ is the final color;
    - $I_D$ is the diffuse intensity, calculated above as the product of the diffuse coefficient and attenuated intensity;
    - $C_D$ is the diffuse reflection color of the surface (e.g. solid color for each object, or something fancy like using the surface normal as color);
    - $C_L$ is the light color (i.e. a chosen solid color defined in light structure above).
  - Here is an example using a dim, red-orange light; the surface color is solid white on the left, and uses the normals on the right (because of its nice gradient):

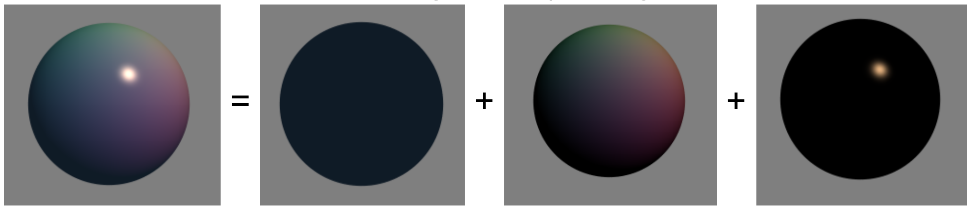
4. ***Phong reflectance***: For slightly more realistic lighting, we will improve upon the Lambertian model by adding *specular highlights*.  The *Phong reflectance* model includes the Lambertian model, but adds a small spot, or highlight, based on the visibility of a reflected ray of light.  This model is sometimes referred to as the *plastic shader* because it makes surfaces look like glossy plastic.
   - ***Specular intensity***: The specular intensity is slightly different from diffuse intensity. It has the form: $I_S = k_S^\alpha$
     - $I_S$ is the specular intensity, the most relevant contribution of the Phong reflectance model;
       - $k_S$ is the specular coefficient, calculated as the alignment of the reflected light with the viewer (see *specular coefficient*);
       - $\alpha$ is the highlight exponent, which determines the focus of the highlight (see *highlight exponent*).
   - ***Specular coefficient***: The specular coefficient is the alignment of the viewer's line of sight to the surface, with the light reflected off of the surface, broken down here:
     - $\hat{V}$ is the *view vector*, the unit vector *from* the surface *towards* the viewer (the viewer's position could be the equivalent of the ray's origin), calculated in a similar fashion as the light vector: $\hat{V} = \text{normalize}(P_{viewer} - P_{surface})$
     - $\hat{R}$ is the *reflected light vector* about the normal: $\hat{R} = \text{reflect}(-\hat{L}, \hat{N})$
     - The specular coefficient formula is: $k_S = \hat{V} \cdot \hat{R}$
       - Note that negative values imply lighting the inside of the surface, much like with the diffuse coefficient; therefore, we have a similar clamped coefficient: $k_S = \max(0, \hat{V} \cdot \hat{R})$
     - Here is the specular coefficient assigned to all color channels:



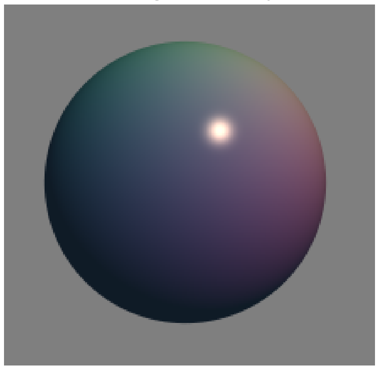
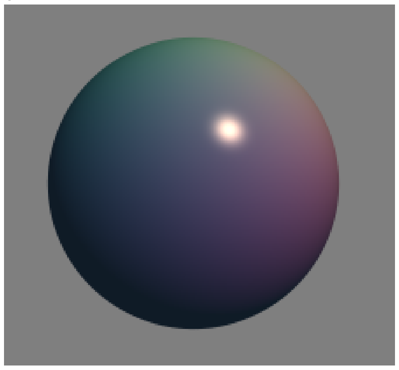
- ***Highlight exponent***: The dot product on its own is not enough to produce a good highlight; we must adjust its size. Since the coefficient is a value between 0 and 1, raising it by some positive exponent will yield a smaller number, thereby reducing its effect; the visual result of this is a more focused highlight. Therefore we apply some positive exponent to the specular coefficient to get the final specular intensity:
  $I_S = k_S^\alpha$
  - $\alpha$ is the highlight exponent, some positive power (*hint: powers of two*).
  - Here is the specular coefficient raised by different powers (top left $\alpha = 4$; top right $\alpha = 16$; bottom left $\alpha = 64$; bottom right $\alpha = 256$; note how the highlight becomes more focused as the power increases):
    
- ***Final color***: The final Phong color is slightly different from the Lambertian color: it is the sum of the *diffuse*, *specular* and *ambient* lighting components:
  $C = I_A C_A + (I_D C_D + I_S C_S) C_L$
  - $C$ is the final color;
    - $I_A$ is the global ambient intensity, some scaling factor for ambient light;
    - $C_A$ is the global ambient color, some global solid color influencing the entire scene;
    - $I_D$ is the light's diffuse intensity (Lambertian reflectance);
    - $C_D$ is the diffuse reflection color of the surface (e.g. solid color of the surface);
    - $I_S$ is the light's specular intensity (Phong or Blinn-Phong reflectance);
    - $C_S$ is the specular reflection color of the surface (can be different from the surface color);
    - $C_L$ is the light's color.
  - Here is the final result, using a red-orange light, surface normal for diffuse color, white for specular color and a bluish tint for ambient:



- Here is the summation using color (Phong = ambient + diffuse + specular):

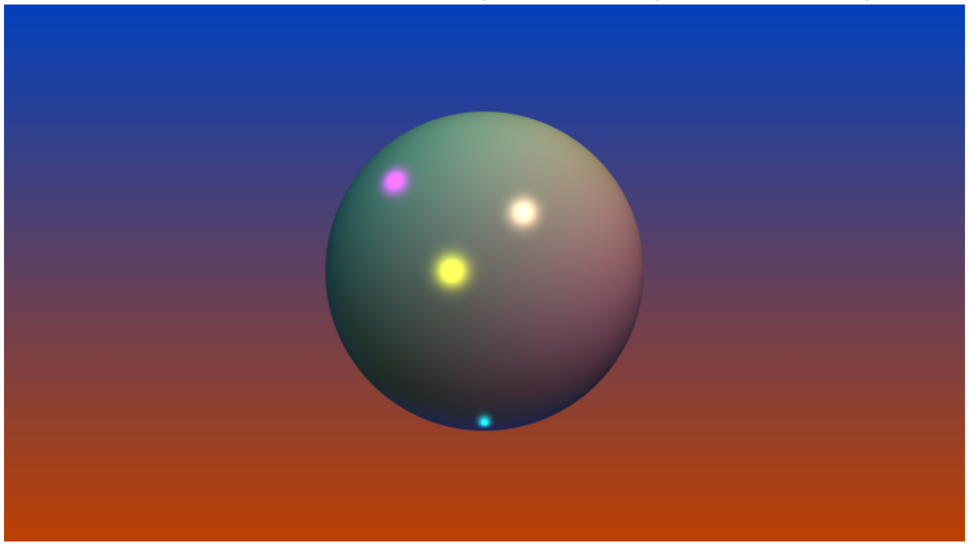


5. ***Blinn-Phong reflectance***: The differences between Phong and Blinn-Phong are subtle, but the latter will typically produce a sharper, more oval-shaped highlight.
   - ***Modified specular intensity***: The Blinn-Phong model is slightly different from the Phong model in that it uses a modified formula for the specular coefficient and exponent: $I_S = {k'_S}^{\alpha'}$
   - ***Halfway vector***: The specular coefficient for the Blinn-Phong model is: $k'_S = \hat{N} \cdot \hat{H}$
     - The difference here is the *halfway vector*, which is the average of the light and view vectors: $\hat{H} = \mathrm{normalize}(\hat{L} + \hat{V})$
   - ***Higher exponent***: The highlight exponent is usually increased for the Blinn-Phong model. A common formula is: $\alpha' = 4\alpha$
   - ***Final color***: Blinn-Phong uses the same formula as Phong using the modified specular coefficient.
     - Here is the Blinn-Phong model (left) compared with the Phong model (right); there is a slight difference in the highlight here (for a better understanding, try comparing them yourself):

6. ***Multiple lights***: The total light influencing the color at any point on the surface, is the sum of all of the lights' influences. Instantiate multiple lights in your implementation and calculate the following sum:

- The final color is the sum of all diffuse and specular reflections (light-dependent) and the ambient color (not light-dependent): $C = I_A C_A + \sum_{i=0}^{n-1} (I_{D_i} C_D + I_{S_i} C_S) C_{L_i}$
  - $C$ is the final color;
    - $I_A$ is the global ambient intensity (not light-dependent);
    - $C_A$ is the global ambient color (not light-dependent);
    - $i$ is the index of the current light;
    - $n$ is the light count;
    - $I_{D_i}$ is the current light's diffuse intensity (Lambertian reflectance);
    - $C_D$ is the diffuse reflection color of the surface (not light-dependent);
    - $I_{S_i}$ is the current light's specular intensity (Phong or Blinn-Phong reflectance);
    - $C_S$ is the specular reflection color of the surface (not light-dependent);
    - $C_{L_i}$ is the current light's color.
  - Here is the final scene with 4 lights and a gradient background:



7. ***Optimization***: Look for ways to optimize any and all of the above equations for ***maximum rendering power***!
   - *Clarification*: This is indeed one of the assignment's expectations, which is why we talk about it in class.  Do 1-6 first, then revisit all of your code to remove redundant operations, organize, etc.  This may take multiple reviews of your code and is also why it is recommended that you ***work in pairs***.  Bring your questions to class.

**Bonus**:
You are encouraged to complete one or more of the following bonus opportunities (rewards listed):

- ***Animate lights (+1)***: Have all of the lights move in fun and interesting patterns around the target objects.
- ***Spot lights (+1)***: Simulate multiple *spot lights* in the scene.  You will need to do some additional reading.
- ***Non-photorealistic lighting model (+1)*:** Implement *cel shading* or *Gooch shading*. You will need to do some additional reading.

**Coding Standards:**
You are required to mind the following standards (penalties listed):

- ***Reminder: You may be referencing others' ideas and borrowing their code. Credit and provide a link to source materials (books, websites, forums, etc.) in your work wherever code from there is borrowed, and credit your instructor for the starter framework, even if it is adapted, modified or reworked (failure to cite sources and claiming source materials as one's own results in an instant final grade of F in the course)***. Recall that borrowed material, even when cited, is not your own and will not be counted for grades; therefore you must ensure that your assignment includes some of your own contributions and substantial modification from what is provided in the book. ***This principle applies to all evaluations***.
- ***Reminder: You must use version control consistently (zero on organization)***. Even though you are using Shadertoy, a platform which allows you to save your work online, you must frequently commit and back up your work using version control. In your repository, create a new text file for each tab used in Shadertoy (e.g. 'Image') and copy your code there. This will also help you track your progress developing your shaders. Remember to work on a new branch for each assignment.
- ***The assignment must be completed using [Shadertoy](https://www.shadertoy.com/) (https://www.shadertoy.com/) (zero on assignment)***. No other platforms will be permitted for this assignment.
- ***Do not reference uniforms in functions other than 'mainImage' (-1 per instance)***: Use the 'mainImage' function provided to call your effect functions. Any uniform value to be used in a function must be passed as a parameter when the function is called from 'mainImage'.
- ***Use the most efficient methods (-1 per inefficient/unnecessary practices)***: Your goal is to implement optimized and thoughtful shader code instead of just implementing the demo as-is. Use inefficient functions and methods sparingly and out of necessity (including but not limited to conditionals, square roots, etc.). Put some thought into everything you do and figure out if there are more efficient ways to do it.
- ***Every section, block and line of code must be commented (-1 per ambiguous section/block/line)***. Clearly state the intent, the 'why' behind each section, block and even line (or every few related lines) of code. This is to demonstrate that you can relate what you are doing to the subject matter.
- ***Add author information to the top of each code file (-1 for each omission)***. If you have a license, include the boiler plate template (fill it in with your own info) and add a separate block with: 1) the name and purpose of the file; and 2) a list of contributors and what they did.

|  |  |
|---|---|
| **Points** | 5 |
| **Submitting** | a text entry box or a website url |

| Due | For | Available from | Until |
|---|---|---|---|
| - | Everyone | - | - |

**GraphicsAnimation-Master-Range**

| Criteria | Ratings | | | Pts |
|---|---|---|---|---|
| IMPLEMENTATION: Architecture & Design<br>Practical knowledge of C/C++/API/framework programming, engineering and architecture within the provided framework or engine. | **1 to >0.5 pts**<br>**Full points**<br>Strong evidence of efficient and functional C/C++/API/framework code implemented for this assignment; architecture, design and structure are largely both efficient and functional. | **0.5 to >0.0 pts**<br>**Half points**<br>Mild evidence of efficient and functional C/C++/API/framework code implemented for this assignment; architecture, design and structure are largely either efficient or functional. | **0 pts**<br>**Zero points**<br>Weak evidence of efficient and functional C/C++/API/framework code implemented for this assignment; architecture, design and structure are largely neither efficient nor functional. | 1 pts |
| IMPLEMENTATION: Content & Material<br>Practical knowledge of content relevant to the discipline and course (e.g. shaders and effects for graphics, animation algorithms and techniques, etc.). | **1 to >0.5 pts**<br>**Full points**<br>Strong evidence of efficient and functional course- and discipline-specific algorithms and techniques implemented for this assignment; discipline-relevant algorithms and techniques are largely both efficient and functional. | **0.5 to >0.0 pts**<br>**Half points**<br>Mild evidence of efficient and functional course- and discipline-specific algorithms and techniques implemented for this assignment; discipline-relevant algorithms and techniques are largely either efficient or functional. | **0 pts**<br>**Zero points**<br>Weak evidence of efficient and functional course- and discipline-specific algorithms and techniques implemented for this assignment; discipline-relevant algorithms and techniques are largely neither efficient nor functional. | 1 pts |
| DEMONSTRATION: Presentation & Walkthrough<br>Live presentation and walkthrough of code, implementation, contributions, etc. | **1 to >0.5 pts**<br>**Full points**<br>Strong evidence of accuracy and confidence in a live walkthrough of code discussing requirements and high-level contributions; walkthrough is largely both accurate and confident. | **0.5 to >0.0 pts**<br>**Half points**<br>Mild evidence of accuracy and confidence in a live walkthrough of code discussing requirements and high-level contributions; walkthrough is largely either accurate or confident. | **0 pts**<br>**Zero points**<br>Weak evidence of accuracy and confidence in a live walkthrough of code discussing requirements and high-level contributions; walkthrough is largely neither accurate nor confident. | 1 pts |
| DEMONSTRATION: Product & Output<br>Live showing and explanation of final working implementation, product and/or outputs. | **1 to >0.5 pts**<br>**Full points**<br>Strong evidence of correct and stable final product that runs as expected; end result is largely both correct and stable. | **0.5 to >0.0 pts**<br>**Half points**<br>Mild evidence of correct and stable final product that runs as expected; end result is largely either correct or stable. | **0 pts**<br>**Zero points**<br>Weak evidence of correct and stable final product that runs as expected; end result is largely neither correct nor stable. | 1 pts |

| Criteria | Ratings | | | Pts |
|---|---|---|---|---|
| ORGANIZATION: Documentation & Management<br>Overall developer communication practices, such as thorough documentation and use of version control. | **1 to >0.5 pts**<br>**Full points**<br>Strong evidence of thorough code documentation and commenting, and consistent organization and management with version control; project is largely both documented and organized. | **0.5 to >0.0 pts**<br>**Half points**<br>Mild evidence of thorough code documentation and commenting, and consistent organization and management with version control; project is largely either documented or organized. | **0 pts**<br>**Zero points**<br>Weak evidence of thorough code documentation and commenting, and consistent organization and management with version control; project is largely neither documented nor organized. | 1 pts |
| BONUSES<br>Bonus points may be awarded for extra credit contributions. | **0 pts**<br>**Points awarded**<br>If score is positive, points were awarded for extra credit contributions (see comments). | **0 pts**<br>**Zero points** | | 0 pts |
| PENALTIES<br>Penalty points may be deducted for coding standard violations. | **0 pts**<br>**Points deducted**<br>If score is negative, points were deducted for coding standard violations (see comments). | **0 pts**<br>**Zero points** | | 0 pts |
| | | | | Total Points: 5 |