Overview: my plan is to first go through the work in ”Ray Tracing in a Weekend”. This book provides an introduction to Ray Tracing by walking the reader through creating a basic ray tracer written in C++. I chose to use this tutorial because it is cited on nVidia’s developer blogs. Once I have written a basic ray tracer in C++ I will implement bounding volume hierarchies and lighting using Peter Shirley’s next book ”Ray Tracing: the Next Week.” By the end of the quarter my goal is to have a ray tracer application that can draw the Cornell Box.

The vec3 class: My first goal is to write a basic vec3 class. I chose to do this myself for this project rather than use the glm::vec3 class because I want a better understanding of some C++ principles that I may be rusty on. I will write my code so that I can easily replace my vec3 class with the glm::vec3 class.

I make heavy use of the ”inline” keyword because the operations in vec3 are very small. The amount of time to retrieve the operation from data memory exceeds the amount of time needed to send the function to the CPU. While the functions will not be called frequently in my program, it is reasonable that a larger program would call all these functions frequently (https://isocpp.org/wiki/faq/inline-functions). I use ”pass-by-const-reference” because I do not want to remind my future self that these functions should not modify the value of the arguments and because I do not want to pass copies of vec3 every time I do vector addition. This could get expensive.

```
class vec3
{
  public:
  //default constructor does nothing
  vec3() {}
  //constructor initializes array of 3 doubles
  vec3(double e0, double e1, double e2) { e[0] = e0; e[1] = e1; e[2] = e2; }
  //getter for x coordinate
  inline double x() const { return e[0]; }
  //getter for y coordinate
  inline double y() const { return e[1]; }
  //getter for z coordinate
  inline double z() const { return e[2]; }
  //getter for r coordinate
  inline double r() const { return e[0]; }
  //getter for g coordinate
  inline double g() const { return e[1]; }
  //getter for b coordinate
  inline double b() const { return e[2]; }
  //Multiplying by 1 does nothing
  inline const vec3& operator+() const { return *this; }
  //Multiplying by -1 negates the vector
  inline vec3 operator-() const { return vec3(-e[0], -e[1], -e[2]); }
  //Bracket accessor returns i'th element
  inline double operator[](int i) const { return e[i]; }
  //Bracket accessor returns a reference to the i'th element
  inline double& operator[](int i) { return e[i]; }
  //+= operator overload (ex v1 += v2;)
  inline vec3& operator+=(const vec3 &v2);
  //-= operator overload (ex v1 -= v2;)
  inline vec3& operator-=(const vec3 &v2);
  //*= operator overload (ex v1 *= v2;)
  inline vec3& operator*=(const vec3 &v2);
  // /= operator overload (ex v1 /= v2;)
  inline vec3& operator/=(const vec3 &v2);
  //*= operator overload (ex v1 *= 3.14;)
  inline vec3& operator*=(const double t);
  ///= operator overload (ex v1 /= 3.14;)
  inline vec3& operator/=(const double t);
  //Returns a unit vector pointing in the direction of the original vector.
  inline vec3 unitize(vec3 v) { return v / v.length(); }
  //Returns the length of the vector.
  inline double length() const { return sqrt(e[0] * e[0] + e[1] * e[1] + e[2] * e[2]); }
  //Returns the length^2 of the original vector
  inline double squared_length() const { return e[0] * e[0] + e[1] * e[1] + e[2] * e[2];
    }
  //Holds the data for the vector: x = e[0] , y = e[1], z = e[2]
  double e[3];
  ~vec3();
};
```

Listing 1: vec3 Class

In addition to the above, I have defined vec3 addition, multiplication, division and substraction. I have also defined the cross product and the dot product. This completes the vec3 class.