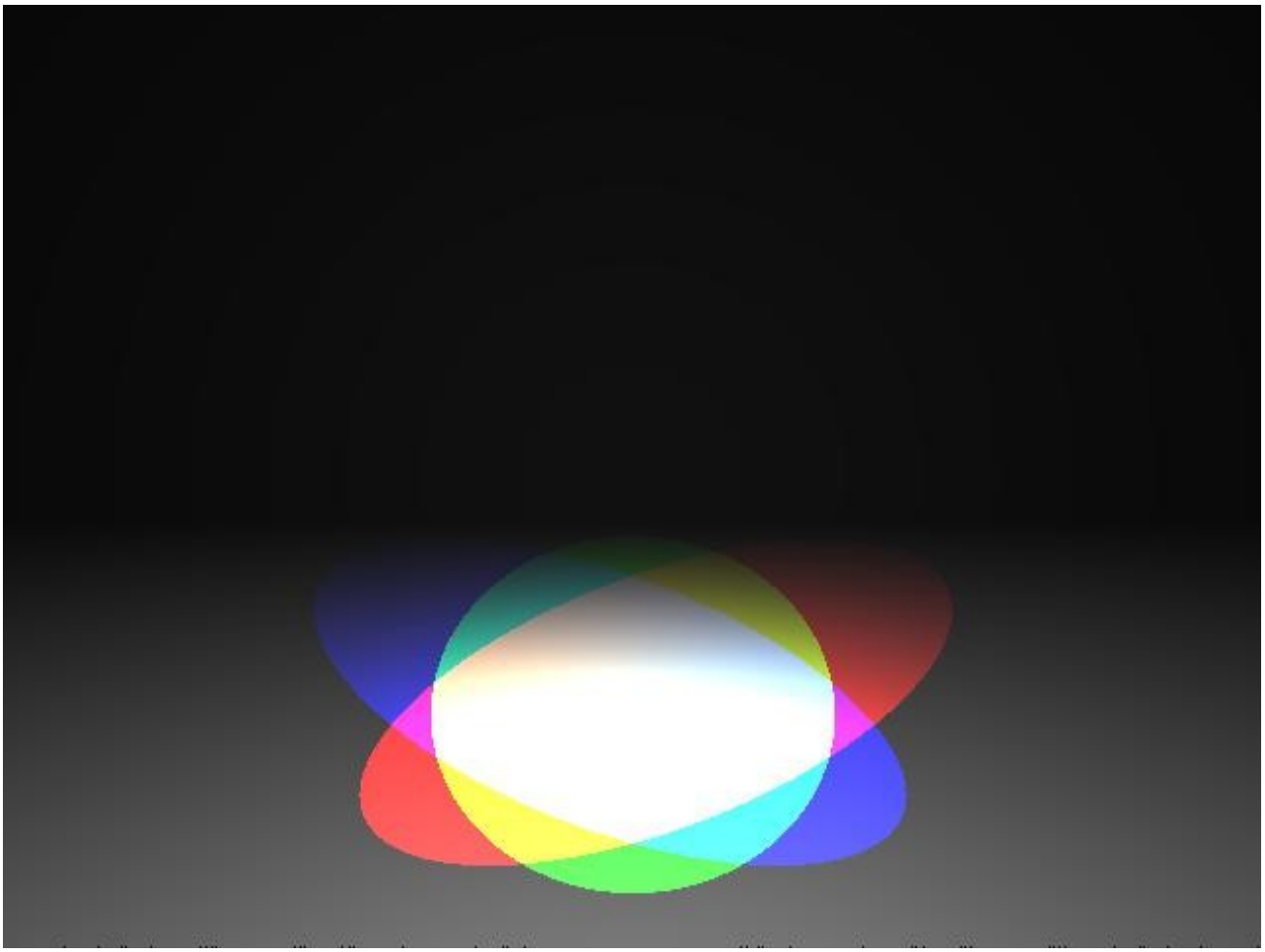
Spotlights

The *light* object that we have been using radiates light in all directions.

A *spotlight* can be thought of as lying at the base of a *cone* and illuminating only that area which is visible from the base of the cone.



Definining a *spotlight* object requires two additional items of information beyond that required for an omnidirectional light:

The direction the spotlight is pointing
The cosine of the angle defining the width of the cone

```
typedef struct spotlight_type{
   double  direction[3];      /* Cone centerline vector    */
   double  theta;             /* Half-width of cone in deg */
   double  costheta;          /* precomputed cos(theta)    */
}  spotlight_t;
```

It turns out to be much easier for a human to *aim* the spotlight if the human is allowed to specify a point on the centerline of the spotlight instead of the direction it points. The direction is needed for visibility computations and can be computed as:

```
vec_diff3(spot->center, spot->hits, spot->direction);
```

```
16              spotlight
4   4   3       center
0   32   0      emissivity
4   -4   -4     a point on the floor that the centerline hits
20.0            cone's half-width theta in degrees
```

Loading the spotlight data

Although it seems easy to "hack" the spotlight code into the existing light loader.

```
if (objtype == SPOTLIGHT)
{
   pcount = scanf("%lf %lf %lf",
                  new->hits,
                  new->hits + 1,
                  new->hits + 2);
   fgets(buf, 256, in);
   pcount += scanf("%lf" ,  &theta);

  Compute cos(theta) and save in spotlight structure.
  Recall cos() wants radians not degrees.

}
```

The theory of *you-touched-it-you-broke-it* says ts better not to do that. Instead, the spotlight should be treated as a derived class of the *light.*

The *spotlight_init* function()

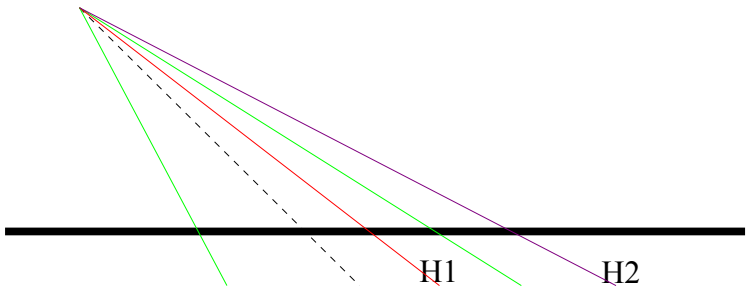
The *spotlight_init()* function is analogous to *fplane_init()*.

```
/**/
obj_t   *spotlight_init(FILE *in, int     objtype)
{
   obj_t        *obj;
   light_t      *light;
   spotlight_t  *spot;
   int          pcount = 0;
   double       theta;
   double       hits[3];

   obj = light_init(in, objtype);
   if (obj == NULL)
      return(obj);

   light = (light_t *)obj->priv;
   spot = malloc(sizeof(spotlight_t));

   light->priv = (void *)spot;
   pcount = vl_get3(in, hits);
   pcount += vl_get1(in, &theta);

   if (pcount != 4)
      return(0);

   Convert hits[3] to spot->direction.

   Compute cos(theta) and save in spotlight structure.
   Recall cos() wants radians not degrees.

   obj->illum_check = spot_visible;

     return(obj);
```

Adjustments to lighting computations

A spotlight can illuminate the *hitloc* if and only if a vector from the *center* of the spotlight to the *hitloc* lies inside the spot cone. Therefore it is necessary to incorporate such a test in the *process_light()* procedure of the *diffuse illumination* module.

In the diagram below the dashed line is the spotlight centerline and the green lines delimit the spot cone. The *hitloc* lies inside the spot cone if and only the angle between the centerline vector and a vector from the center of the spotlight to the hitpoint is less than *theta* the angle that defined the halfwidth of the spot cone. The point H1 is illuminated by the spotlight but H2 is not.



Therefore to determine if a *hitloc* is illuminated:

1. Compute a *unit* vector *from* the center of the spotlight *to* the *hitloc*
2. Take the dot product of this vector with a *unit* vector in the direction of the centerline.
3. If this value is *greater than* the *costheta* value previously computed, the *hitloc* is illuminated.

Implementing the illumination test

As usual it would be possible to hack the test into the middle of process light.

```
/* If the object is a spotlight ensure the hitpoint is within */
/* the cone.                                                  */

   if (lobj->objtype == SPOTLIGHT)
   {
     if hitobj->hitloc is not in the spot cone
          return(0);
   }

   obj = (obj_t *)lst->head;
   while (obj != NULL)
```

As usual this would be a bad idea. A better idea is to use a "virtual" or polymorphic function that could be used to test not only *spotlights* but also *projectors* and other conceivable light forms. In the C language it is common to use an *if it exists, call it* approach for implementing such functions. In this way an omni-directional light doesn't need to provide a *default* function that always returns *0.* At the end of *spotlight_init()* the *obj->illum_check* pointer was set to point to the function *spot_visible.*

```
/* If the light is a directional light such as a spotlight or */
/* a projector it may have a special visibility function      */

   if (lobj->illum_check)
   {
      if (lobj->illum_check(lobj, hitobj->hitloc))
         return(0);
   }
```

However, it can also be argued that doing it this way is *dangerous* because some later program maintainer might not realize that *illum_check()* functions were optional and might (fatally) attempt to invoke one via a NULL pointer.