Using a texture as a source of illumination

Consider the traditional color slide projector. A bright light is placed behind the translucent slide and directed toward the screen. We can consider the light source as casting rays which pass through the slide and illuminate the screen. In passing through the color slide the ray is filtered in such a way that it takes on the color of the slide element through which it passes.

We can model this process by placing a light source behind a translucent textured plane. The illumination it provides is model as follows. For each *hitpoint* on a reflective object a ray is fired from the illumination source toward the hit point. If the ray hits the textured plane, then the projector potentially illuminates the hitpoint. The emissivity of the projector at the hitpoint is modeled as the component-wise product of the emissivity of the light source with the texel through which the ray passes.

The *projplane_type*

```
typedef struct obj_type
{
   struct  obj_type  *next;      /* Next object in list         */
   int     objid;                /* Numeric serial # for debug  */
   int     objtype;              /* obj code (14 -> Plane, etc )*/
   double  (*hits)(double *base, double *dir, struct obj_type *);

   ...
   void    (*getemiss)(struct obj_type *, double *);
   void    (*getloc)(struct obj_type *, double *);
   int     (*vischeck)(struct obj_type *, double *);
}obj_t;

typedef struct plane_type
{
   double  normal[3];
   double  point[3];
   double  ndotq;
   void    *priv;         /* Data for specialized types  */
}  plane_t;

typedef struct fplane_type
{
   double  xdir[3];
   double  size[2];
   double  lasthit[2];
   void    *priv;           /* data for specialized types */
} fplane_t;

typedef struct texplane_type
{
   int         texmode;     /* 1 -> fit and 2-> tile     */
   char        texname[40]; /* name of .ppm file         */
   texture_t  *texture;     /* pointer to texture struct */
   void        *priv        /* Data for specialized types */
}  texplane_t;

typedef struct projplane_type
{
   double      flen;              /* focal len             */
   double      center[3];         /* light source          */
}  projplane_t;
```

The projplane inheritance structure is shown above Relevant aspects of the definition are noted below.

- The *vischeck* function determines if a ray from the *location* of the projector light source to a *hit location* passes through the textured plane. If not the projector doesn't illuminate the object.
- The *getemiss()* function returns the component-wise product of the emissivity of the projectors light source with the texel through which the ray from the light source to the hit location passes.
- The *getloc()* function copys the center of the light source to another vector location.

Loading a projector plane

The protector plane management functions *projplane_init, projplane_dump, projplane_vischeck, projplane_getloc,* and *projplane_emiss* should reside in a new module called projplane.c

```
obj_t *projplane_load(FILE *in, int objtype)
{
   projplane_t *ppln;
   texplane_t  *texp;
   fplane_t    *fp;
   plane_t     *p;
   obj_t       *obj;
   char        buf[256];

   obj = texplane_load(in, objtype);
   if(obj == NULL)
      return NULL;
```

recovere pointer to tex_plane_t structure
malloc new projplane and link it to the tex_plane_t

set hit function pointer to the hit function for finite planes
set getamb/getspec/getdiff function pointers to function that copy the value in the material structure within the object structure (because lights font have functions to calculate these values)
set getemiss, getloc, and vischeck function pointers

(we do not really need the getloc function for the ray tracer to work) Its job is simple enough that you might never notice the lack

read in focal length

It should be noted at this point, that **light** objects do not read in ambient, diffuse and specular lighting values, but instead only read in emissivity values.

projplane_t functions

Creating a new projplane_t creates instances of *object_t, plane_t, fplane_t,* and *texplane_t.* The *projplane_t* overrides only the *getemiss()* and *vischeck()* and *getloc()* with new function pointers.

```
int projplane_vischeck(obj_t *obj, double *dir)
```

- Ask *fplane_hits()* if a ray fired from center of the projector plane in direction *dir* hits underlying *fplane*.
- If so, return 0 else return 1

```
}
```

```
void projplane_emiss(obj_t *obj, double *emiss)
{
   double texel[3];
```

- Ask *texture_map()* to return the *texel* that the ray from the light location to the *hitloc* passes through. Note that *this works only because of the work done in vischeck().* The value stored in *lasthit* when *vischeck()* called fplane_hits() will be used here.
- Store component-wise product of *texel* and *emissivity* in return value *emiss*

```
}
```

Patches to *process_light*

```
int process_light(
list_t *lst,           /* List of all objects            */
obj_t  *hitobj,        /* The object hit by the ray      */
obj_t  *lobj,          /* the current light source       */
double *ivec)          /* [r, g, b] intensity vector     */
{
```

if the hitobj occludes itself
return

if we have a vischeck function (which is only true for some lights)
ask vischeck to determine if the light can illuminate the hitloc
if not return

find_closest_object() along a ray from hitloc to the center of the light
if one exists and is closer than the light // the light is occluded by the object
return

*compute the illumination and add it to *pix;* ***(which could be modularized by a call to a function similar to the one below.)***

```
void illuminate(
obj_t   *hitobj,       /* object that was hit by the ray */
obj_t   *light,        /* a visible light object         */
double  dist,          /* length from the center of the light to the hitlocation */
double  cos,           /* angle used in self occlusion test */
double intensity[3])   /* where to add intensity         */
{
```

Arriving at this point means the light does illuminate
object. Ask hitobj->getdiff() to compute diffuse reflectivity
.....
Ask light->getemiss() to return the emissivity of the light
.....
Multiply componentwise the diffuse reflectivity by
the emissivity of the light.
.....
Scale the resulting diffuse reflectivy by cos/dist
.....
*Add scaled value to *pixel.*

}