The textured plane

In an earlier assignment you wrote a program which loaded a *ppm* image and then mapped it onto a rectangular plane. Such an image is commonly called a *texture* and the process of mapping it onto a planar surface is called *texture mapping*.

We can incorporate this approach into the raytracer. The *textured plane* is simply a *finite plane* onto which a texture has been mapped. Both *tiled* and *fit* mode may be employed as shown in the following example entitled: “Philosophers at Dinner”.

The image above includes 4 textured planes. For the curtains, the marble floor, the frame on which the photograph is mounteda and the photograph. The texture is mapped in *tiled* mode for the frame and the floor. The curtains and the photograph itself are mapped in *fit* mode. Note specular refection of the wall and photograph show in the marble floor.

The textured plane definition

The textured plane definition requires two items of information beyond that of the finite rectangular plane: the name of the file containing the texture and the mapping mode. Mapping mode *1* means stretch the texture to *fit* the plane. Mapping mode *2* means repeatedly *tile* the texture.

```
17              textured plane
6.0 6.0 6.0     amb, diffuse, spec
4.0 4.0 4.0     amb, diffuse, spec
0.0 0.0 0.0     amb, diffuse, spec
0 0 1           normal
-8 -6 -4.2      point

1 0 0           xdir
24 18           width height

marble.ppm      name of the texture file
2               tile mode (1 means fit mode)
```

The *tex_plane* inheritance structure

```
typedef struct obj_type{
   ...
   void    *priv;

} obj_t;

typedef struct plane_type{
   double  normal[3];
   double  point[3];
   double  ndotq;         /* Used multiple times        */
   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 */
}  texplane_t;
```

Loading a textured plane

The textured plane management functions *texplane_load, texplane_dump, texplane_amb, texplane_diff,* and *texplane_spec* should reside in a new module called *texplane.c*

```
obj_t *texplane_load(FILE *in, list_t  *lst, int objtype) {
   texplane_t *texp;
   fplane_t   *fp;
   plane_t    *p;
   obj_t      *obj;
   int        pcount;
   char       buf[256];

/* Create fplane_t, plane_t, and obj_t structures */

   obj = fplane_load(in, lst, objtype);
   if (obj == NULL)
      return(NULL);

   recover pointer to fplane_t structure
   malloc new texplane_t and link it to fplane_t

   set obj->hits to fplane_hits

   set obj->getamb/getdiff to texplane_getamb/getdiff

   read in texture file name and texture file mode.

   if (texture_load(texp))
      return(NULL);

   return(obj);
}
```

Loading the texture

Each texture will be represented by a structure of the following type

```
typedef struct texture_type {
   int            size[2];    /* x dim – y dim */
   unsigned char *texbuf;     /* pixmap buffer */
}  texture_t;
```

The *texture_load()* and *texture_map()* functions should live in a new module named *texture.c*

```
int texture_load(texplane_t *tp) {
```

attempt to fopen() the image file and return `exit failure' on error

read and parse the .ppm header returning `exit failure' if it's not a P6 .ppm image

malloc a texture_t structure and link it to the texplane_t

save texture dimensions in the texture_t

malloc a texture buffer to hold the pixel data and save its address in the texture_t

read in the pixel data from the pixmap.

return success or failure

```
}
```

Determining the *diffuse* pixel color of the *textured* plane.

As was the case with the *tiled* plane, the real action occurs in *getamb/getdiff*. We will describe the action of the *getdiff()* function *texplane_diff()*.

The basic idea is that the intensity returned will be the *product* of the object's reflectivity with the texel that maps to the hit point. The *texture_map()* function in *texture.c* is responsible for determining what *texel* maps to the hit point.

```
/**/
void texplane_diff(obj_t *obj, double *value){
   plane_t  *pln = (plane_t *)obj->priv;
   fplane_t *fp  = (fplane_t *)pln->priv;

   double texel[3];

   texture_map(fp, texel);
   compute product of reflectivity and texel

}
```

The *texture_map* function

This mission of this function is to determine the *texel* which maps to the most recent hit point on the object.   In general this would appear to be a difficult thing to do but, as with the finite plane, its not so hard if the textured plane is based at (0, 0, 0), has *x* direction (1, 0, 0), and normal (0, 0, 1).

Thus, to simulate this condition, we can simply use the “newhit” data that was computed in *fplane_hits* and stored in *fp->lasthit[]* at the time the *hit* was found:

```
fp->lasthit[0] = newhit[0];
fp->lasthit[1] = newhit[1];
```

Acquiring the value of a texel is easy when the *x* and *y* offsets of the hit are expressed as a fractional percentage of the *x* and *y* dimensions of the texture.   This is the main mission of the *texture_map()* function.

```
int texture_map(fplane_t *fp, double *texel){
```

recover pointers to the texplane_t and the texture_t

if the texture mode is TEX_FIT {

compute the fractional x-offset, xfrac, of the texel as fp->lasthit[0] / fp->size[0]

pass xfrac and yfrac to texel_get()

}

else / mode is TEX_TILE */ {*

This mode is slightly more complicated because the size of the texture must be considered. There are two possible ways to do this: (1) convert the size of the texture to world coordinates or (2) map the fp->lasthit to pixel coordinates.

Since we already have a map_pix_to_world() function it might seem easier to use approach (1) but I recommend using approach (2) because it corresponds better to the approach suggested in you modern art maker. To do this you will need to build a map_world_to_pix() function.

Assuming you have done map_world_to_pix(fp->lasthit, pixhit);

Then xfrac = (double)(pixhit[0] % texture->size[0])/ texture_size[0];

pass xfrac and yfrac to texel_get()

}

The *texel_get()* function

This functions mission is to copy the contents of the texel at fractional position (xrel, yrel) into the texel passed as a parameter.

```
/*
 *      xrel- relative offset of [0.0, 1.0)
 *      yrel – within the texture
*/
void texel_get(texture_t *tex, double xrel, double yrel,
               double texel[3]){
   unsigned char *texloc;
   int xtex;      /* x coordinate of the texel */
   iny ytex;      /* y coordinate of the texel */

   multiply x rel and y rel by respective texture dimensions getting xtex and ytex;
   ensure 0 <= xtex < x texture size and 0 < = ytex < y texture size

   compute offset, texloc, of the (ytex, xtex) in texture in the usual way

   texel[0] = *texloc / 255.0;
   texel[1] = *(texloc+1) / 255.0;
   texel[2] = *(texloc+2) / 255.0;

}
```

Mapping world to pixel coordinates

You will note that information in the *proj_t* is required to map between pixel and world coordinates. Furthermore the *proj_t* structure is not passed down the diffuse and ambient lighting paths.

There are several ways (all somewhat ugly) to work around this:

(1) pass a pointer to the *proj_t* from *mk_image()* through *raytrace()* through *diffuse_illumination()....*

(2) in module *proj.c* where the mapping functions should reside (I am aware some have wandered off into other files) create a *static proj_t *p;* that is *not* in the body of any function. Initialize it to point to the *proj_t* structure in *projection_init()*. Then use this pointer to access the *proj_t* structure in *map_world_to_pix()*.

(3) Add *pix_x_size* and *pix_y_size* elements to the *fplane_t* structure. Then hack a patch into *model_load()* (which does hold a reference to the *proj_t),* Each time a *texplane_t* is loaded, the patch can compute the values of *pix_x_size* and *pix_y_size.*

The third method is the one I used in my C++ version since I already had to have the *case* structure in place anyway.

```
case TEX_PLANE:
/* fprintf(stderr, "Loading finite pln\n"); */
  texplane  = new texplane_t(in, out, objtype, objid);
  texplane->set_pixsize(proj->pix_x_size, proj->pix_y_size);
  list->add(texplane);
  break;
```

In my C version I decided that option (2) was the easiest and least intrusive.