Assignment 2
CS 405/805-001: Computer Graphics
Instructor: Xue Dong Yang
Tuesday, October 2, 2012
Due Date: Tuesday, October 23, 2012

In this assignment, you are to implement a simple ray-tracing algorithm outlined below:

```
main ()
{
       Initialize the global data structures;

       for (i=0; i<ROWS; i++)              // scan through each row
         for (j=0; j<COLS; j++) {          // scan through each column

             Construct the ray, V, started from the CenterOfProjection
              and passing through the pixel (i, j);

             if ((c = ray-tracing( V ) != nil)     // if the ray intersects
                                                   // with an object
                 image[i][j] = c;          // save the returned shading
                                           // value into the image buffer.
         }

       Output the final image;
}


int ray-tracing( L )
{
       P = ray-objects-intersection( L );  // return the nearest intersection
                                           // point if any

       If (P != nil) {
           C = shading( P );
           return ( C );
       } else
           return( nil );
}
```

The main computations are in the following three functions:

```
// Construction of ray V
// Input:     pixel index (i, j) in the screen coordinates
// Output:    V = (P0, V0) (for parametric ray equation P = P0 + V0*t)
//             in the world coordinates.
```

```
// Note: V is only a logical symbol for the ray in the algorithm. The real
// representation of V is P0 and V0.
Void ray_construction(int i, int j, float P0[3], float V0[3])
{
       map (j, i) in the screen coordinates to (xc, yc) in the camera
              coordinates;

       transform the origin (0.0, 0.0, 0.0) of the camera coordinates to P0
              in the world coordinates using the transformation matrix Mcw;
       transform the point (xc, yc, f) on the impage plane in the camera
              coordinates to P1 in the world coordinates using the
              transformation matrix Mcw;
       V0 = P1  -  P0;
       Normalize V0m into unit length;
}



// Ray-Object Intersection
// Input:     ray  -  P0, V0
// Output:    the nearest intersection point P[3] if found, along with
//                   N[3], the surface normal at that point, and
//                   kd, the diffuse reflection coefficient of the surface.
// Note: In a general system, the objects should be stored a list structure.
// A loop will scan through each object in the list. The nearest intersection
// point is found. In our case, we will have only two hard-coded objects:
// a sphere and a polygon.So, this part is “hard-coded”.
Void ray_object_intersection(float P0[3], float V0[3])
{
       t1 = ray-sphere-intersection(float P0[3], float V0[3],
              struct sphere S, float N1[3], float kd1);
       t2 = ray-polygon-intersection(float P0[3], float V0[3],
              struct polygon PL, float N2[3], float kd2);
       if (t1 == nil and t2 == nil)
              return nil;
       else if (t2 == nil)
              return (t1, N1, kd1);
       else if (t1 == nil)
              return(t2, N2, kd2);
       else if (t1 < t2)
              return (t1, N1, kd1);
       else
              return (t2, n2, kd2);

}

// Shading:
// Input:     P[3]  -  point position
//            N[3]  -  surface normal at that point
//            kd  -  diffuse reflection coefficient of the surface
```

```
// Output:    C  -  shading value
Int shading(float P[3], float N[3], float kd)
{
       L = LRP  -  P;         // LRP is the light position
       normalize L to unit length;
       C = Ip * kd * (N * L);       // where (N * L) is dot-product
}
```

Global Data Structure:

It includes, but not limited to the followings:

- The cameral model – VRP, VPN, VUP
- The light position – LRP
- The transformation matrices: Mwc, Mcw
- Image buffer image[ROWS][COLS];

Input Model:

Ideally, all objects should be defined in a script data file. In the initialization stage, the objects are read into the program and stored in a list structure. In this assignment, we will have only two objects – a sphere and a polygon. To simplify this part of processing, you are allowed to hard-code the objects, together with the global data structure into a header file (e.g. "model.h").

The camera model and light information are usually also specified in the input script file. You are also allowed to hard-code them in the header file. However, the transformation matrices Mwc and Mcw should computed in the initialization stage.

I will provide a sample "model.h" file. You should test your program with this data file. You are then required to:

- Modify the camera model in the "model.h" to generate a picture from a different view; and
- Modify the polygon position in the "model.h" to generate a picture with a slightly different scene.

---------- END ----------