Assignment 3
CS 405/805-001: Computer Graphics
Instructor: Xue Dong Yang
Tuesday, October 23, 2012
Due Date: Tuesday, November 13, 2012

You are to implement the basic volume rendering algorithm (ray-tracing type) based on Marc Levoy's paper "Display of Surfaces from Volume Data."

The CT data set can be copied from my directory by the following steps:

- Log into your Unix account either on "venus" or "hercules";
- Go to your local working directory, e.g.:
  `cd cs805/ <CR>`
- Copy the CT data set from my directory to your local directory:
  `cp ~yang/DATA/cs805/smallHead.den . <CR>`

You should have file named "smallHead.den" with a size 2097152 bytes in your local directory now. If you have difficulty to copy this file, you may bring a USB memory to my office to copy it.

This is a binary data file containing 128X128X128 unsigned char type (i.e. 8 bit per voxel) in Z-Y-X order.

A simple ray-tracing volume rendering algorithm is outlined below:

/* This is a C-like Pseudo Language Algorithm.
It is NOT a complete C Program. */

```
int main ()
{
        FILE *infid, *outfid; /* input and output file id's */
        int n;
        int i, j;

        /* Load the CT data into the array */
        if ((infid = fopen("smallHed.den", "rb")) == NULL) {
                printf("Open CT DATA File Error.\n");
                exit(1);
        }
        for (i=0; i<SLCS; i++) { /* Read one slice at a time. */
                n = fread(&CT[i][0][0], sizeof(char), ROWS*COLS, infid);
                if (n < ROWS*COLS*sizeof(char)) {
                        printf("Read CT data slice %d error.\n", i);
                        exit(1);
                }
        }
```

```
        /* Compute the shading volume */
        Compute-shading-volume();

        Initialize the global data structures; // You need to write this part.

        /* ================================================
               The Main Ray-Tracing Volume Rendering Part.
        /* ================================================

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

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

              float ts[2];    // for storing the intersection points t0 and t1.
              n = ray-box-intersection(V, ts); // n is the number of
                                               // intersections found.
              if (n == 2)     // only if you have two intersections
                  image[i][j] = volume-ray-tracing(V, ts);
          }

          /* Save the output image */

        outfid = fopen( "outimage.raw" , "wb");
        n = fwrite(out_img, sizeof(char), IMG_ROWS*IMG_COLS, outfid);
        if (n < IMG_ROWS*IMG_COLS*sizeof(char)) {
               printf("Write output image error.\n");
               exit(1);
        }
        fclose(infid);
        fclose(outfid);
        exit(0);
}
```

The main computations are in the following functions:

```
/* Compute the shading values at each voxel:
        - Compute the partial derivatives at each voxel;
        - Normalize the partial derivative into a unit-length vector;
        - Apply a simplified illumination model for shading:
                        I = Ip*(N*L)
*/
Void Compute-shading-volume()
{
        // You need to write the body.

}
```

```
// Main Volume Ray-Tracing Function:
// Input: V = (P0, V0) (for parametric ray equation P(t) = P0 + V0*t)
//        ts[0] and ts[1] are two points on the ray. Assuming ts[0] < ts[1]
// Output: the integrated shading value along the ray between ts[0] and ts[1]

int volume-ray-tracing( V, ts[2])
{
       float Dt = 20.0;      // the interval for sampling along the ray
       float C = 0.0;        // for accumulating the shading value
       float T = 1.0;        // for accumulating the transparency

        /* Marching through the CT volume from t0 to t1
           by step size Dt.
        */
        for (t=t0; t<=t1; t += Dt) {       // front-to-back order

           /* Compute the 3D coordinates of the current
              sample position in the volume:
              x = x(t);
              y = y(t);
              z = z(t);
           */


           /* Obtain the shading value C and opacity value A
              from the shading volume and CT volume, respectively,
              by using tri-linear interpolation. */


           /* Accumulate the shading values in the front-to-back order.

              Note: You will accumulate the transparency. This value
              can be used in the for-loop for early termination.
           */

       }
}


// 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-Box Intersection
// Input:      ray - P0, V0
// Output:     n - the number of intersections found
//                    = 0, no intersection
//                    = 1, one intersection
//                    = 2, two intersections
//                    It will be very rare to find more two intersections.
//                    In this case, you can set it to 0.
//             ts[2] stores the found intersections.
//             if n = 2, you should have ts[0] < ts[1].
//                    kd, the diffuse reflection coefficient of the surface.
int ray_box_intersection(float P0[3], float V0[3], float ts[2])
{
        int n = 0;

         /* Intersect the line with the CT cube to find
            two intersection points, t0 and t1, corresponding
            to the entry and exit points respectively.

            Notes: the six sides of the CT cube are defined
            in the world coordinates by:
               x = 0;
               x = 127;
               y = 0;
               y = 127;
               z = 0;
               z = 127.
         */

        /* Example: for side x = 0
               x(t) = P0[0] + V[0] * t = 0.0
               t can be solved using the above equation.
               Then, substitute t into the following two equations
                 to compute y and z:
               y(t) = p0[1] + V0[1] * t
               z(t) = P0[2] + V0[2] * t
               The y and z need to be checked against the rectangular
```

```
            boundaries (this is so much easier than the general
            boundary checking that you did in the Assignment 2):
          If (y > 0 && y < ROWS && z > 0 && z < SLCS) then
                Save the t value into ts[] and update n.
        */

        /* do the remaining five cases */

        Return( n );
}

// Another function needed is the tri-linear interpolation function.
// I leave it to you to write.
```

Global Data Structure:

It includes, but not limited to the followings:

- The cameral model – VRP, VPN, VUP
- The light direction – LPN (assuming parallel light)
- The transformation matrices: Mwc, Mcw
- Image buffer image[IMG_ROWS][IMG_COLS];
- Input volume CT[SLCS][ROWS][COLS];
- Shading volume SHADING[SLCS][ROWS][COLS];

Input Model:

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.

Similar to Assignment 2, you are allowed to hard-code some of the above global values in a header file (e.g. "mymodel.h"). You may modify your previous file. I will also provide a sample "mymodel.h" file. You should test your program with this data file. You are then required to:

- Modify the camera model in the "mymodel.h" to generate a picture from a different view.

----------------------------------- **END** --------------------------------------