

# cs805 Assignment 2

Ray Shulang Lei

200253624

Department of Computer Science

University of Regina

October 26, 2012

## **Abstract**

This assignment is written in literate programming style, generated by noweb, rendered by LaTeX, and compiled by clang++ with c++11 standard.

assignment paper is at latex/as2.pdf

c++ programs are at src/\*

binary executable for OS X 10.8 is inside bin

# 1 function implementation

```
<<src/util.cpp>>=
#include "util.h"
#include <cmath>

//pixel iterator for img panel.
ImagePanel foreach_pixel_exec(ImagePanel img, std::function<int(Ray)> ray_func) {
    int i = 0;
    for (auto& pixel: img) { //foreach pixel in empty_img
        //using to_2d function to get x,y camera coordinates
        auto cam_xy = to_2d(i);

        //construct Ray
        Ray ray = ray_construction(cam_xy[0], cam_xy[1]);
        pixel = ray_func(ray);
        i++;
    }
    return img;
}

//ray constructor
Ray ray_construction(int x, int y) {
    //calculate x unit
    double x_delta = (xmax-xmin) / IMG_X;
    double y_delta = (ymax-ymin) / IMG_Y;

    //calculate the point on img panel with world coordinate
    double x_ = xmax - x_delta * x;
    double y_ = ymax - y_delta * y;

    //get vector v0. it is trival that VRP is p0
    Point p0 = VRP;
    Point p1_ = {x_, y_, focal};
    Point p1 = mul(Mcw, p1_);
    Vector v0_ = {p1[0] - p0[0],
        p1[1] - p0[1],
```

```

    p1[2] - p0[2]};
    Vector v0 = normalize(v0_);

    if ((x==0 ) || (x==511)) {
        std::cout<<"img: x:"<<x<<" , y:"<<y;
        //std::cout<<"p0: x:"<<VRP[0]<<" , y:"<<VRP[1]<<" , z:"<<VRP[2]<<std::endl;
        //std::cout<<"p0: x:"<<p0[0]<<" , y:"<<p0[1]<<" , z:"<<p0[2]<<std::endl;
        std::cout<<"p1_: x:"<<p1_[0]<<" , y:"<<p1_[1]<<" , z:"<<p1_[2]<<"=====";
        std::cout<<"p1: x:"<<p1[0]<<" , y:"<<p1[1]<<" , z:"<<p1[2]<<"=====";
        std::cout<<"v0: x:"<<v0[0]<<" , y:"<<v0[1]<<" , z:"<<v0[2]<<"=====";
        std::cout<<std::endl;
    }

    return { p0[0], p0[1], p0[2],
            v0[0], v0[1], v0[2]};
}

//initialize img panel to all 0s
ImagePanel init_img_panel(ImagePanel img) {
    for (auto& pixel: img) { //foreach pixel in empty_img
        pixel = 0;
    }
    return img;
}

//translate ray equation to an 0~255 shading value
int ray_tracing(Ray ray) {
    Intersection p = ray_objects_intersection(ray);
    return shading(p);
}

//calculate the ray object intersection point
Intersection ray_objects_intersection(Ray ray) {
    auto sphere_hit = ray_sphere_intersection(ray);
    auto polygon_hit = ray_polygon_intersection(ray);
    if (sphere_hit.kd < 0 && polygon_hit.kd < 0) {
        return {-1,-1,-1,
                -1,-1,-1,

```

```

        -1.0};
    } else if (polygon_hit.kd < 0) {
        return sphere_hit;
    } else if (sphere_hit.kd < 0) {
        return polygon_hit;
    } else if (closer(sphere_hit.intersection, polygon_hit.intersection, {0,0,0})) {
        return sphere_hit;
    } else {
        return polygon_hit;
    }
}

```

```

Intersection ray_sphere_intersection(Ray ray) {
    /*
    std::cout<<"ref: "<<ray.ref[0]<<" "
        <<ray.ref[1]<<" "
        <<ray.ref[2]<<" direction: "
        <<ray.direction[0]<<" "
        <<ray.direction[1]<<" "
        <<ray.direction[2]<<std::endl;
    std::cout<<"obj1: "<<obj1.x<<std::endl;
    */
    return {-1,-1,-1,
        -1,-1,-1,
        -1.0};
}

```

```

Intersection ray_polygon_intersection(Ray ray) {
    return {-1,-1,-1,
        -1,-1,-1,
        -1.0};
}

```

```

//calculate shading value from 0~255 accordingly to intersection info
int shading(Intersection p) {
    if (p.kd < 0) {
        return -1;
    }
}

```

```

    return 255;
}

//=====helpers=====

//prints a matrix
void pmatrix(std::string str, Matrix m) {
    std::cout<<str<<std::endl;
    for (auto row : m) {
        for (auto num : row) {
            std::cout<<std::setw (10);
            std::cout<<num;
        }
        std::cout<<std::endl;
    }
    std::cout<<std::endl;
}

//get transformation matrix
Matrix get_T(Point vrp) {
    Row r1 = {1, 0, 0, -vrp[0]};
    Row r2 = {0, 1, 0, -vrp[1]};
    Row r3 = {0, 0, 1, -vrp[2]};
    Row r4 = {0, 0, 0, 1};
    return {r1, r2, r3, r4};
}

//get inverse transformation matrix
Matrix get_Ti(Point vrp) {
    Row r1 = {1, 0, 0, vrp[0]};
    Row r2 = {0, 1, 0, vrp[1]};
    Row r3 = {0, 0, 1, vrp[2]};
    Row r4 = {0, 0, 0, 1};
    return {r1, r2, r3, r4};
}

//get rotation matrix

```

```

Matrix get_R(Point vrp, Vector vpn, Vector vup) {
    //first get the translation matrix from world to view
    //auto mt = get_T(vrp);

    //we can see vpn_ and vup_ as vectors. such that we can apply them to get_uvn fu
    auto uvn = get_uvn(vpn, vup);
    //finally construct our roation matrix using method 2 on class notes
    Row r1 = { uvn[0][0], uvn[0][1], uvn[0][2], 0 };
    Row r2 = { uvn[1][0], uvn[1][1], uvn[1][2], 0 };
    Row r3 = { uvn[2][0], uvn[2][1], uvn[2][2], 0 };
    Row r4 = { 0, 0, 0, 1 };
    return { r1, r2, r3, r4 };
}

//get inverse rotation matrix
Matrix get_Ri(Point vrp, Vector vpn, Vector vup) {
    Matrix m = get_R(vrp, vpn, vup);
    Row r1 = { m[0][0], m[1][0], m[2][0], m[3][0] };
    Row r2 = { m[0][1], m[1][1], m[2][1], m[3][1] };
    Row r3 = { m[0][2], m[1][2], m[2][2], m[3][2] };
    Row r4 = { m[0][3], m[1][3], m[2][3], m[3][3] };
    return {r1,r2,r3,r4};
}

//world to camera
Matrix get_M(Point vrp, Vector vpn, Vector vup) {
    return mul(get_R(vrp, vpn, vup), get_T(vrp));
}

//camera to world
Matrix get_Mi(Point vrp, Vector vpn, Vector vup) {
    return mul(get_Ti(vrp), get_Ri(vrp, vpn, vup));
}

//matrix multiplication
Point mul(Matrix m, Point x) {
    return mul(x, m);
}

```

```

}

Point mul(Point x, Matrix m) {
    double w = m[3][0] * x[0]
        + m[3][1] * x[1]
        + m[3][2] * x[2]
        + m[3][3];
    return {(x[0]*m[0][0]+x[1]*m[0][1]+x[2]*m[0][2]+m[0][3])/w,
        (x[0]*m[1][0]+x[1]*m[1][1]+x[2]*m[1][2]+m[1][3])/w,
        (x[0]*m[2][0]+x[1]*m[2][1]+x[2]*m[2][2]+m[2][3])/w};
}

Matrix mul(Matrix m, Matrix n) {
    Row r1 = {m[0][0]*n[0][0]+m[0][1]*n[1][0]+m[0][2]*n[2][0]+m[0][3]*n[3][0],
        m[0][0]*n[0][1]+m[0][1]*n[1][1]+m[0][2]*n[2][1]+m[0][3]*n[3][1],
        m[0][0]*n[0][2]+m[0][1]*n[1][2]+m[0][2]*n[2][2]+m[0][3]*n[3][2],
        m[0][0]*n[0][3]+m[0][1]*n[1][3]+m[0][2]*n[2][3]+m[0][3]*n[3][3]};
    Row r2 = {m[1][0]*n[0][0]+m[1][1]*n[1][0]+m[1][2]*n[2][0]+m[1][3]*n[3][0],
        m[1][0]*n[0][1]+m[1][1]*n[1][1]+m[1][2]*n[2][1]+m[1][3]*n[3][1],
        m[1][0]*n[0][2]+m[1][1]*n[1][2]+m[1][2]*n[2][2]+m[1][3]*n[3][2],
        m[1][0]*n[0][3]+m[1][1]*n[1][3]+m[1][2]*n[2][3]+m[1][3]*n[3][3]};
    Row r3 = {m[2][0]*n[0][0]+m[2][1]*n[1][0]+m[2][2]*n[2][0]+m[2][3]*n[3][0],
        m[2][0]*n[0][1]+m[2][1]*n[1][1]+m[2][2]*n[2][1]+m[2][3]*n[3][1],
        m[2][0]*n[0][2]+m[2][1]*n[1][2]+m[2][2]*n[2][2]+m[2][3]*n[3][2],
        m[2][0]*n[0][3]+m[2][1]*n[1][3]+m[2][2]*n[2][3]+m[2][3]*n[3][3]};
    Row r4 = {m[3][0]*n[0][0]+m[3][1]*n[1][0]+m[3][2]*n[2][0]+m[3][3]*n[3][0],
        m[3][0]*n[0][1]+m[3][1]*n[1][1]+m[3][2]*n[2][1]+m[3][3]*n[3][1],
        m[3][0]*n[0][2]+m[3][1]*n[1][2]+m[3][2]*n[2][2]+m[3][3]*n[3][2],
        m[3][0]*n[0][3]+m[3][1]*n[1][3]+m[3][2]*n[2][3]+m[3][3]*n[3][3]};
    return {r1,r2,r3,r4};
}

Row mul(Row x, Matrix m) {
    return {x[0]*m[0][0]+x[1]*m[0][1]+x[2]*m[0][2]+x[3]*m[0][3],
        x[0]*m[1][0]+x[1]*m[1][1]+x[2]*m[1][2]+x[3]*m[1][3],
        x[0]*m[2][0]+x[1]*m[2][1]+x[2]*m[2][2]+x[3]*m[2][3],
        x[0]*m[3][0]+x[1]*m[3][1]+x[2]*m[3][2]+x[3]*m[3][3]};
}

```

```

Row mul(Matrix m, Row x) {
    return mul(x, m);
}

//return if p1 is closer to p0 than p2
bool closer(Point p1, Point p2, Point p0) {
    double d1 = (p1[0] - p0[0])+(p1[1] - p0[1])+(p1[2] - p0[2]);
    double d2 = (p2[0] - p0[0])+(p2[1] - p0[1])+(p2[2] - p0[2]);
    return d1 < d2;
}

//Translate 2D array index of row column to 1D index.
//Notice that x, or column index, starts with 0.
//If return value is -1 then there is an out-of-bounce error.
int to_1d(int x, int y) {
    if (x >= IMG_X || x < 0)
        return -1;
    if (y >= IMG_Y || y < 0)
        return -1;
    return (IMG_Y*y + x);
}

//Translate 1d array index to 2d
std::array<int, 2> to_2d(int x) {
    if (x >= (IMG_X*IMG_Y) || x < 0) {
        return {-1, -1};
    }
    int y_ = x / IMG_X;
    int x_ = x % IMG_X;
    return {x_, y_};
}

//prints the img panel
void print_img_panel(ImagePanel img) {
    std::cout<<std::endl;
    for (auto& pixel : img) {
        std::cout<<pixel<<" ";
    }
}

```



```

    }
    std::cout<<std::endl<<"Array size: "<<img.size()<<std::endl;
}

//get u,v,n from two non-collinear vectors
UVN get_uvn(Vector V1, Vector V2) {

    //get n, which is just normalized V1
    Vector n = normalize(V1);

    //get u, which is normalized V2 x V1
    Vector u = normalize(cross_product(V2, V1));

    //get v, which is normalized n x u
    Vector v = normalize(cross_product(n, u));

    return {u,v,n};
}

//normalize a Vector
Vector normalize(Vector x) {
    return { x[0]/get_length(x),
            x[1]/get_length(x),
            x[2]/get_length(x) };
}

//calculates cross product of two Vectors
Vector cross_product(Vector x, Vector y) {
    return { x[1]*y[2] - x[2]*y[1],
            x[2]*y[0] - x[0]*y[2],
            x[0]*y[1] - x[1]*y[0] };
}

//calculates length of a Vector
double get_length(Vector x) {
    return sqrt(pow(x[0],2)+pow(x[1],2)+pow(x[2],2));
}

```

@

## 2 header

Here is an header file for typedefs and function declarations.

```
<<src/util.h>>=
#ifndef UTIL_H
#define UTIL_H

//define preprocessing vars
#define IMG_X 512
#define IMG_Y 512
#define IMG_LEN ( IMG_X * IMG_Y )
/* definition of the image buffer */
#define ROWS IMG_Y
#define COLS IMG_X

#include <array>
#include <functional>
#include <iostream>
#include <iomanip>

//types
typedef std::array<int, IMG_LEN> ImagePanel;
typedef std::array<double, 3> Point;
typedef std::array<double, 3> Vector;
typedef std::array<Vector, 3> UVN;
typedef struct {
    Point intersection; /* intersection point */
    Vector normal; /* intersection polygon normal vector */
    double kd; /* diffuse reflection coefficient of the surface */
} Intersection;
typedef struct {
    Point ref; /* reference point, where the ray is from */
    Vector direction; /* ray direction */
```

```

} Ray;
typedef std::array<double, 4> Row;
typedef std::array<Row, 4> Matrix;
typedef struct {
double x, y, z; /* center of the circle */
double radius; /* radius of the circle */
double kd; /* diffuse reflection coefficient */
} SPHERE;
typedef struct {
double v[4][3]; /* list of vertices */
double N[3]; /* normal of the polygon */
double kd; /* diffuse reflection coefficient */
} POLY4;

//functions
ImagePanel foreach_pixel_exec(ImagePanel, std::function<int(Ray)>);
ImagePanel init_img_panel(ImagePanel);
int ray_tracing(Ray);
Intersection ray_objects_intersection(Ray);
int shading(Intersection);
Intersection ray_sphere_intersection(Ray);
Intersection ray_polygon_intersection(Ray);
Ray ray_construction(int, int);

//helper functions
Point mul(Point, Matrix);
Point mul(Matrix, Point);
Matrix mul(Matrix, Matrix);
Row mul(Row, Matrix);
Row mul(Matrix, Row);
int to_1d(int, int);
std::array<int, 2> to_2d(int);
void print_img_panel(ImagePanel);
void pmatrix(std::string, Matrix);
bool closer(Point, Point, Point);
UVN get_uvn(Vector V1, Vector V2);
Matrix get_T(Point);
Matrix get_Ti(Point);

```

```

Matrix get_R(Point, Vector, Vector);
Matrix get_Ri(Point, Vector, Vector);
Matrix get_M(Point, Vector, Vector);
Matrix get_Mi(Point, Vector, Vector);
double get_length(Vector);
Vector cross_product(Vector, Vector);
Vector normalize(Vector);

```

```

//global vars
extern Matrix Mwc;
extern Matrix Rwc;
extern Matrix Twc;
extern Matrix Mcw;
extern Matrix Rcw;
extern Matrix Tcw;
extern Matrix Mwl;
extern Matrix Mlw;
extern double xmin;
extern double ymin;
extern double xmax;
extern double ymax;
extern Point VRP;
extern Vector VPN;
extern Vector VUP;
extern double focal;
extern Point LRP;
extern double Ip;
extern SPHERE obj1;
extern POLY4 obj2;
#endif
@

```

### 3 main funciton

```

<<src/main.cpp>>=
#include <iostream>

```

```

#include "util.h"

/* create a spherical object */
SPHERE obj1 = {1.0, 1.0, 1.0, /* center of the circle */
  1.0, /* radius of the circle */
  0.75}; /* diffuse reflection coefficient */

/* create a polygon object */
POLY4 obj2 = { 0.0, 0.0, 0.0, /* v0 */
  0.0, 0.0, 2.0, /* v1 */
  2.0, 0.0, 2.0, /* v2 */
  2.0, 0.0, 0.0, /* v3 */
  0.0, 1.0, 0.0, /* normal of the polygon */
  0.8}; /* diffuse reflection coefficient */

//unsigned char img[ROWS][COLS];

/* definition of window on the image plane in the camera coordinates */
/* They are used in mapping (j, i) in the screen coordinates into */
/* (x, y) on the image plane in the camera coordinates */
/* The window size used here simulates the 35 mm film. */
double xmin = 0.0175;
double ymin = -0.0175;
double xmax = -0.0175;
double ymax = 0.0175;

/* definition of the camera parameters */
Point VRP = {1.0, 2.0, 3.5};
Vector VPN = {0.0, -1.0, -2.5};
Vector VUP = {0.0, 1.0, 0.0};

double focal = 0.05; /* focal length simulating 50 mm lens */

/* definition of light source */
Point LRP = {-10.0, 10.0, 2.0}; /* light position */
double Ip = 200.0; /* intensity of the point light source */

```

```

/* Transformation from the world to the camera coordinates */
Matrix Mwc = get_M(VRP, VPN, VUP);
Matrix Rwc = get_R(VRP, VPN, VUP);
Matrix Twc = get_T(VRP);
/* Transformation from the camera to the world coordinates */
Matrix Mcw = get_Mi(VRP, VPN, VUP);
Matrix Rcw = get_Ri(VRP, VPN, VUP);
Matrix Tcw = get_Ti(VRP);
/* Transformation from the world to light coordinates */
Matrix Mwl = get_T(LRP);
/* Transformation from the light to the world coordinates */
Matrix Mlw = get_Ti(LRP);

int main () {
    ImagePanel img;
    img = init_img_panel(img);
    img = foreach_pixel_exec(img, ray_tracing);
    //print_img_panel(img);

    //tests
    Point vrp = {6.0, 10.0, -5.0};
    Vector vpn = {-6.0, -9.0, 5.0};
    Vector vup = {0.0, 1.0, 0.0};
    auto uvn = get_uvn(vpn, vup);
    std::cout<<"get_uvn function:"<<std::endl;
    for (auto vecotr : uvn) { //for each Vecotr in uvn
        for (auto num : vecotr) { //for each number in Vecotr
            std::cout<<num<<',';
        }
        std::cout<<std::endl;
    }

    Matrix mwc = get_M(vrp, vpn, vup);
    Matrix mcw = get_Mi(vrp, vpn, vup);
    Matrix twc = get_T(vrp);
    Matrix tcw = get_Ti(vrp);
    Matrix rwc = get_R(vrp, vpn, vup);
    Matrix rcw = get_Ri(vrp, vpn, vup);

```

```

pmatrix("mwc:", mwc);
pmatrix("twc:", twc);
pmatrix("rwc:", rwc);
pmatrix("mcw:", mcw);
pmatrix("tcw:", tcw);
pmatrix("rcw:", rcw);

pmatrix("Mcw:", Mcw);

std::cout<<"to_1d function, expected to be 512:"<<std::endl;
std::cout<<to_1d(0, 1)<<std::endl;

std::cout<<"to_2d function, expected to be 0, 1:"<<std::endl;
std::cout<<to_2d(512) [0]<<std::endl;
std::cout<<to_2d(512) [1]<<std::endl;

std::cout<<"to_1d function, expected to be 513:"<<std::endl;
std::cout<<to_1d(1, 1)<<std::endl;

std::cout<<"to_2d function, expected to be 1,1:"<<std::endl;
std::cout<<to_2d(513) [0]<<std::endl;
std::cout<<to_2d(513) [1]<<std::endl;

std::cout<<"to_1d function, expected to be 1023:"<<std::endl;
std::cout<<to_1d(511, 1)<<std::endl;

std::cout<<"to_2d function, expected to be 511,1:"<<std::endl;
std::cout<<to_2d(1023) [0]<<std::endl;
std::cout<<to_2d(1023) [1]<<std::endl;

std::cout<<"to_1d function, expected to be -1:"<<std::endl;
std::cout<<to_1d(512, 1)<<std::endl;

std::cout<<"to_2d function, expected to be -1,-1:"<<std::endl;
std::cout<<to_2d(512*512) [0]<<std::endl;
std::cout<<to_2d(512*512) [1]<<std::endl;

std::cout<<"closer function, expected to be 1 and 0:"<<std::endl;

```

```

std::cout<<closer({1,1,1},{2,2,2},{0,0,0})<<std::endl;
std::cout<<closer({3,3,3},{2,2,2},{0,0,0})<<std::endl;

std::cout<<"mul function: point * matrix"<<std::endl;
Point a = {3,3,3};
std::cout<<mul(a, Mcw)[0]<<std::endl;
std::cout<<mul(Mcw, a)[1]<<std::endl;
std::cout<<mul(a, Mcw)[2]<<std::endl;

pmatrix("mul funciton: matrix*matrix:", mul(Mwc, Mcw));

std::cout<<"mul function: row * matrix"<<std::endl;
Row b = {3,3,3,3};
std::cout<<mul(b, Mcw)[0]<<std::endl;
std::cout<<mul(Mcw, b)[1]<<std::endl;
std::cout<<mul(b, Mcw)[2]<<std::endl;
std::cout<<mul(b, Mcw)[3]<<std::endl;

return 0;
}
@

```

## 4 compile script

Furthermore, this is the command to link these files. Notice that I am using -std=c++11 flag to enable c++ 11 features. The output binary executable is bin/run

```

<<compile.sh>>=
clang++ -std=c++11 -stdlib=libc++ -o bin/run src/main.cpp src/util.cpp
@

```