cs805 Assignment 2

Ray Shulang Lei
200253624
Department of Computer Science
University of Regina

October 23, 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 <math.h>

//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 ray_construction(int x, int y) {
  //transform VRP to world coordinate
    return {-1,-1,-1,
            -1,-1,-1};
}

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

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

```
//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) {
    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==========

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

@
```

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>

//types
typedef std::array<int, IMG_LEN> ImagePanel;
typedef std::array<float, 3> Point;
typedef std::array<float, 3> Vector;
typedef struct {
Point intersection; /* intersection point */
Vector normal; /* intersection polygon normal vector */
float 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<float, 4> Row;
typedef std::array<Row, 4> Matrix;
/* Definition of the structure for Sphere */
typedef struct {
float x, y, z; /* center of the circle */
float radius; /* radius of the circle */
float kd; /* diffuse reflection coefficient */
} SPHERE;
/* Definition of Polygon with 4 edges */
typedef struct {
float v[4][3]; /* list of vertices */
float N[3]; /* normal of the polygon */
float 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);
int to_1d(int, int);
std::array<int, 2> to_2d(int);
void print_img_panel(ImagePanel);
bool closer(Point, Point, Point);
#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. */
float xmin = 0.0175;
float ymin = -0.0175;
float xmax = -0.0175;
float ymax = 0.0175;


/* definition of the camera parameters */
float VRP[3] = {1.0, 2.0, 3.5};
float VPN[3] = {0.0, -1.0, -2.5};
float VUP[3] = {0.0, 1.0, 0.0};

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


/* definition of light source */
float LPR[3] = {-10.0, 10.0, 2.0}; /* light position */
float Ip = 200.0; /* intensity of the point light source */

/* === transformation matrices (to be constructed) === */

/* Transformation from the world to the camera coordinates */
Matrix Mwc =
{1.0, 0.0, 0.0, 0.0,
 0.0, 1.0, 0.0, 0.0,
 0.0, 0.0, 1.0, 0.0,
 0.0, 0.0, 0.0, 1.0};

/* Transformation from the camera to the world coordinates */
Matrix Mcw =
{1.0, 0.0, 0.0, 0.0,
 0.0, 1.0, 0.0, 0.0,
 0.0, 0.0, 1.0, 0.0,
 0.0, 0.0, 0.0, 1.0};

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

  //unit tests
  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:"<<std::endl;
  std::cout<<mul({3,3,3},Mcw)[0]<<std::endl;
  std::cout<<mul(Mcw, {3,3,3})[1]<<std::endl;
  std::cout<<mul({3,3,3},Mcw)[2]<<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
@
```