4F03 Project

Authors

Name	Student Number	Email	Website
Jamie Counsell	1054209	counsej@mcmaster.ca	jamiecounsell.me
James Priebe	1135001	priebejp@mcmaster.ca	

Description

This C program renders a mandelbulb or mandelbox 3D fractal with optimizations in OpenACC.

Dependencies

- `pgcc` or compiler with support for OpenACC

Installation

- Install the dependencies
- Clone the repository

Operation

To run the program, first run make:

```
$ make clean
$ make [type]
```

Where `type` is one of:

- **`mandelbulb`** - Compute a Mandelbulb fractal
- **`mandelbox`** - Compute a Mandelbox fractal
- **`boxserial`** - Compute a Mandelbox fractal using a serial implementation
- **`bulbserial`** - Compute a Mandelbulb fractal using a serial implementation

Then run the command with the optional runtime flags:

```
$ ./mandel[box, bulb, bulb_serial, box_serial] params.dat [-f n] [-v]
```

where:

- **params.dat** is a file containing the Mandelbulb or Mandelbox parameters.
- **f** - Instruct the program to generate `n` frames. Default is 1 frame.
- **v** - Instruct the program to generate a video when it is complete (calling `genvideo.sh`)

For example, to generate a 7200 frame video at 30FPS (4 minutes) of the mandelbulb:

```
$ ./mandelbulb params.dat -f 7200 -v
```

To generate the mandelbulb given in the assignment, one can use the command:

```
$ make clean; make mandelbulb
$ ./mandelbulb paramsBulb.dat
```

OR the serial version:

```
$ make clean; make bulbserial
$ ./bulbserial paramsBulb.dat
```

The resulting images will be in the frames directory as `00000.bmp`. The filename used in the parameters is not used here to follow convention and ensure this frame can be used in the video.

Speedups

For the first frame of the submitted video, the following times were recorded:

Server	OpenACC	time
tesla	NO	108.16456s
tesla	YES	1.236812s

The server was under heavy load during testing, so future results may vary, but this shows a significant speedup (~87.5x faster with OpenACC than without). The same CPU was used to show speedups related purely to OpenACC acceleration.

Parallelization

The only region that was parallelized was the nested loop in `renderer.cc`. This loop is the program's largest bottleneck and also supports parallelization quite intuitively. OpenACC pragmas were used to identify the region as an OpenACC compute region, as well as transfer the data to the device from the host. The outer loop was explicitly marked as parallel, and other optimizations were left up to PGCC.

Functions called inside the compute region were identified as ACC Routines, and any functions called within such routines were inlined, due to the issue mentioned in class, where variables seem to take on a NULL or somewhat undefined value when they are passed to a function called by a Routine, even if that function is also marked as a Routine.

The data structures were flattened to be more easily passed between methods. To accomodate for this, additional parameters were added to the ACC Routines called inside the compute region. All data (including the flattened parameter structures) was explicity copied to the device using data pragmas before the beginning of the compute region.

Early on, we faced an interesting problem (and a great example of the proper use of the `present_or_copy[in, out]` ACC Methods. When image data (`image`) was marked as `copy`, `copyin`, or `copyout`, the compiler would generate code to reallocate `image` on the device each time the compute region began. This is because it has no way to maintain state across compute regions, and is therefore unable to maintain the pointer to `image` without explicitly checking if it is present first (then reuisng it). By adding the `present_or` prefix, we were able to instruct the compiler to not reallocate the memory, and instead overwrite the existing memory allocated for `image`. Since every pixel in `image` is changed before it is copied out, there are no problems here with risk of using old data.

Since frame parameters were not generated asynchronously, no parallelization was done to compute more than one frame at a time.

Frame Generation

Frames are generated sequentially from an array of `CameraParams` structures. The first image generated is always the same as what is identified in the input parameters. This ensures that the assignment requirements can be properly met with the given `paramsBulb.dat` file. After the first image, the camera rotates around the fractal, slowly decreasing its position in the `z` axis from `1` to `-1` across 7200 frames. The position is computed as follows:

- the `x` coordinate is `cos(frame_number/500)`
- the `y` coordinate is `sin(frame_number/500)`

- the `z` coordinate is `1 - (frame_number/3600)`

This will guide the camera around the object in a circular motion along the `x, y` plane such that it will complete one full rotation every `500*pi` frames. The z value decreases individually from `1` to `-1` between frames `0` and `7200`, respectively. This creates a sort of *spring* path, showcasing all sides of the fractal.

Each iteration, `init3D` is called again to ensure the camera is still facing the center point at `(0,0,0)`.

With the exception of the first frame, each subsequent frame's parameters are generated during the previous frame's position in the loop. That is, the parameters for frame `i` are computed before rendering frame `i-1`. An array of `CameraParams` structures are kept in order to keep track of current and previous configurations. One could add support for rendering multiple frames at once, since the configurations are all available in memory. This would be a reasonable next step, and a good use for something like OpenMP.

Final Result

To compute the final result, the following configuration file (`bulb_params.dat`) was used:

```
# CAMERA
# location x,y,z (7,7,7)
1 0 1
# look at x,y,z
0 0 0
# up vector x,y,z; (0, 1, 0)
0 0 1
# field of view (1)
2.0
# IMAGE
# width height
3840 2160
# detail level, the smaller the more detailed (-3.5)
-3.45
# MANDELBULB
# ignore the first number, 0.
# the second and third numbers are escape (or bailout) time and power
0 4.0 9.0
# ignore the second number; the first number is the max number of iterations
100 0
# COLORING
# type 0 or 1
0
# brightness
1.2
# IMAGE FILE NAME
imageBulb.bmp
```

The command used to generate the result is:

```
$ ./mandelbulb bulb_params.dat -f 7200 -v
```

The URL for the video will be available on the GitHub repository and will be sent via email to the course instructor and TAs.

Source Code

See attached.

Source Code

4F03 Project

The files of most interest are `walk.cc`, `walk.h`, `raymarching.cc`, and `renderer.cc`.

makefile

```makefile
all: mandelbox

clean:
	rm -f *.o mandelbulb mandelbox boxserial bulbserial *~

bulbserial:
	make -f make.serial_bulb

boxserial:
	make -f make.serial_box

mandelbulb:
	make -f make.bulb

mandelbox:
	make -f make.box
```

make.box

```makefile
all: box

CXX      =  pgc++
GPUFLAGS = -fast -acc -ta=tesla,cc35 -Minfo=accel -Minline
FLAGS    = -O3
CFLAGS   = $(FLAGS)
CXXFLAGS = $(GPUFLAGS) -DBOX
LDFLAGS  = $(FLAGS)

PROGRAM_NAME=mandelbox

OBJS =  main.o print.o timing.o savebmp.o getparams.o 3d.o getcolor.o raymarching.o
renderer.o init3D.o walk.o

box: $(OBJS)
	$(CXX) $(CXXFLAGS) -o $(PROGRAM_NAME) $? $(LDFLAGS)

clean:
	rm -f *.o mandelbox *~
```

make.bulb

```makefile
all: bulb

CXX      =  pgc++
GPUFLAGS = -fast -acc -ta=tesla,cc35 -Minfo=accel -Minline
FLAGS    = -O3
CFLAGS   = $(FLAGS)
CXXFLAGS = $(GPUFLAGS) -DBULB
LDFLAGS  = $(FLAGS)

PROGRAM_NAME=mandelbulb30

OBJS =  main.o print.o timing.o savebmp.o getparams.o 3d.o getcolor.o raymarching.o
renderer.o init3D.o walk.o

bulb: $(OBJS)
	$(CXX) $(CXXFLAGS) -o $(PROGRAM_NAME) $? $(LDFLAGS)

clean:
	rm -f *.o mandelbulb *~
```

make.serial_box

```makefile
all: serial

CXX      =  g++
FLAGS    = -O3 -Wall
CXXFLAGS = $(FLAGS) -DBOX
LDFLAGS  = -lm

PROGRAM_NAME=boxserial

OBJS =  main.o walk.o print.o timing.o savebmp.o getparams.o 3d.o getcolor.o raymarc
hing.o renderer.o init3D.o

serial: $(OBJS)
	$(CXX) -o $(PROGRAM_NAME) $? $(CXXFLAGS) $(LDFLAGS)

clean:
	rm -f *.o boxserial *~
```

make.serial_bulb

```makefile
all: serial

CXX      =  g++
FLAGS    = -O3 -Wall
CXXFLAGS = $(FLAGS) -DBULB
LDFLAGS  = -lm

PROGRAM_NAME=bulbserial

OBJS =  main.o walk.o print.o timing.o savebmp.o getparams.o 3d.o getcolor.o raymarc
hing.o renderer.o init3D.o

serial: $(OBJS)
	$(CXX) -o $(PROGRAM_NAME) $? $(CXXFLAGS) $(LDFLAGS)

clean:
	rm -f *.o bulbserial *~
```

3d.cc

```c
/*
   This file is part of the Mandelbox program developed for the course
    CS/SE  Distributed Computer Systems taught by N. Nedialkov in the
    Winter of 2015-2016 at McMaster University.

    Copyright (C) 2015-2016 T. Gwosdz and N. Nedialkov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "camera.h"
#include "3d.h"
#include "vector3d.h"

#ifdef _OPENACC
#include <openacc.h>
#endif


// CHANGES FOR OPENACC
// MultiplyMatrixByVector is now inlined to avoid a calling depth greater than one
// UnProject takes explicit camera parameters rather than passing a structure
// int[3] array has been replaced by a vec3 struct due to problems with OpenACC
//    being unable to privatize the array
// Remaining functions are unchanged



inline void MultiplyMatrixByVector(double *resultvector, const double *matrix, doubl
e *pvector)
{
```

```
  resultvector[0]=matrix[0]*pvector[0]+matrix[4]*pvector[1]+matrix[8]*pvector[2]+matrix[12]*pvector[3];
  resultvector[1]=matrix[1]*pvector[0]+matrix[5]*pvector[1]+matrix[9]*pvector[2]+matrix[13]*pvector[3];
  resultvector[2]=matrix[2]*pvector[0]+matrix[6]*pvector[1]+matrix[10]*pvector[2]+matrix[14]*pvector[3];
  resultvector[3]=matrix[3]*pvector[0]+matrix[7]*pvector[1]+matrix[11]*pvector[2]+matrix[15]*pvector[3];
}

//---------------------------------------------------------------------------------------
//when projection and modelview matricies are static (computed only once, and camera does not move)
#pragma acc routine seq
void UnProject(double winX, double winY, const int viewport[4], const double matInvProjModel[16], vec3 &obj)//double *obj)
{

  //Transformation vectors
  double in[4], out[4];

  //Transformation of normalized coordinates between -1 and 1
  in[0]=(winX-(double)(viewport[0]))/(double)(viewport[2])*2.0-1.0;
  in[1]=(winY-(double)(viewport[1]))/(double)(viewport[3])*2.0-1.0;
  in[2]=2.0-1.0;
  in[3]=1.0;

  //Objects coordinates
  MultiplyMatrixByVector(out, matInvProjModel, in);

  if(out[3]==0.0){
    //return 0;
  }else{

    out[3] = 1.0/out[3];
    obj.x/*[0]*/ = out[0]*out[3];
    obj.y/*[1]*/ = out[1]*out[3];
    obj.z/*[2]*/ = out[2]*out[3];

    //return 1;
  }

}
```

```
// END OPENACC CHANGES

void LoadIdentity(double *matrix){
  matrix[0] = 1.0;
  matrix[1] = 0.0;
  matrix[2] = 0.0;
  matrix[3] = 0.0;

  matrix[4] = 0.0;
  matrix[5] = 1.0;
  matrix[6] = 0.0;
  matrix[7] = 0.0;

  matrix[8] = 0.0;
  matrix[9] = 0.0;
  matrix[10] = 1.0;
  matrix[11] = 0.0;

  matrix[12] = 0.0;
  matrix[13] = 0.0;
  matrix[14] = 0.0;
  matrix[15] = 1.0;
}

//--------------------------------------------------------------------------------------
------
void Perspective(double fov, double aspect, double zNear, double zFar, double *projM
at)
{
  double ymax, xmax;

  ymax = zNear * tan(fov * M_PI / 360.0);
  //ymin = -ymax;
  //xmin = -ymax * aspectRatio;
  xmax = ymax * aspect;
  Frustum(-xmax, xmax, -ymax, ymax, zNear, zFar, projMat);
}

void Frustum(double left, double right, double bottom, double top, double znear, dou
ble zfar, double *matrix)
{
  double temp, temp2, temp3, temp4;
  temp = 2.0 * znear;
  temp2 = right - left;
  temp3 = top - bottom;
```

```
  temp4 = zfar - znear;
  matrix[0] = temp / temp2;
  matrix[1] = 0.0;
  matrix[2] = 0.0;
  matrix[3] = 0.0;
  matrix[4] = 0.0;
  matrix[5] = temp / temp3;
  matrix[6] = 0.0;
  matrix[7] = 0.0;
  matrix[8] = (right + left) / temp2;
  matrix[9] = (top + bottom) / temp3;
  matrix[10] = (-zfar - znear) / temp4;
  matrix[11] = -1.0;
  matrix[12] = 0.0;
  matrix[13] = 0.0;
  matrix[14] = (-temp * zfar) / temp4;
  matrix[15] = 0.0;
}
//-----------------------------------------------------------------------
void LookAt(double *eye, double *target, double *upV, double *modelMatrix)
{
  double forward[3], side[3], up[3];
  double matrix2[16], resultMatrix[16];
  //------------------
  forward[0] = target[0] - eye[0];
  forward[1] = target[1] - eye[1];
  forward[2] = target[2] - eye[2];
  NormalizeVector(forward);
  //------------------
  //Side = forward x up
  ComputeNormalOfPlane(side, forward, upV);
  NormalizeVector(side);
  //------------------
  //Recompute up as: up = side x forward
  ComputeNormalOfPlane(up, side, forward);
  //------------------
  matrix2[0] = side[0];
  matrix2[4] = side[1];
  matrix2[8] = side[2];
  matrix2[12] = 0.0;
  //------------------
  matrix2[1] = up[0];
  matrix2[5] = up[1];
  matrix2[9] = up[2];
  matrix2[13] = 0.0;
```

```
  //-------------------
  matrix2[2] = -forward[0];
  matrix2[6] = -forward[1];
  matrix2[10] = -forward[2];
  matrix2[14] = 0.0;
  //-------------------
  matrix2[3] = matrix2[7] = matrix2[11] = 0.0;
  matrix2[15] = 1.0;
  //-------------------
  MultiplyMatrices(resultMatrix, modelMatrix, matrix2);
  Translate(resultMatrix, -eye[0], -eye[1], -eye[2]);
  //-------------------
  memcpy(modelMatrix, resultMatrix, 16*sizeof(double));
}


void NormalizeVector(double *v)
{
  double m = 1.0/sqrt(v[0]*v[0]+v[1]*v[1]+v[2]*v[2]);
  v[0] *= m;
  v[1] *= m;
  v[2] *= m;
}

void ComputeNormalOfPlane(double *normal, double *v1, double *v2)
{
  normal[0] = v1[1] * v2[2] - v1[2] * v2[1];
  normal[1] = v1[2] * v2[0] - v1[0] * v2[2];
  normal[2] = v1[0] * v2[1] - v1[1] * v2[0];
}

void MultiplyMatrices(double *result, const double *matrix1, const double *matrix2)
{
  result[0]=matrix1[0]*matrix2[0]+
    matrix1[4]*matrix2[1]+
    matrix1[8]*matrix2[2]+
    matrix1[12]*matrix2[3];
  result[4]=matrix1[0]*matrix2[4]+
    matrix1[4]*matrix2[5]+
    matrix1[8]*matrix2[6]+
    matrix1[12]*matrix2[7];
  result[8]=matrix1[0]*matrix2[8]+
    matrix1[4]*matrix2[9]+
    matrix1[8]*matrix2[10]+
    matrix1[12]*matrix2[11];
```

```
    result[12]=matrix1[0]*matrix2[12]+
      matrix1[4]*matrix2[13]+
      matrix1[8]*matrix2[14]+
      matrix1[12]*matrix2[15];
    result[1]=matrix1[1]*matrix2[0]+
      matrix1[5]*matrix2[1]+
      matrix1[9]*matrix2[2]+
      matrix1[13]*matrix2[3];
    result[5]=matrix1[1]*matrix2[4]+
      matrix1[5]*matrix2[5]+
      matrix1[9]*matrix2[6]+
      matrix1[13]*matrix2[7];
    result[9]=matrix1[1]*matrix2[8]+
      matrix1[5]*matrix2[9]+
      matrix1[9]*matrix2[10]+
      matrix1[13]*matrix2[11];
    result[13]=matrix1[1]*matrix2[12]+
      matrix1[5]*matrix2[13]+
      matrix1[9]*matrix2[14]+
      matrix1[13]*matrix2[15];
    result[2]=matrix1[2]*matrix2[0]+
      matrix1[6]*matrix2[1]+
      matrix1[10]*matrix2[2]+
      matrix1[14]*matrix2[3];
    result[6]=matrix1[2]*matrix2[4]+
      matrix1[6]*matrix2[5]+
      matrix1[10]*matrix2[6]+
      matrix1[14]*matrix2[7];
    result[10]=matrix1[2]*matrix2[8]+
      matrix1[6]*matrix2[9]+
      matrix1[10]*matrix2[10]+
      matrix1[14]*matrix2[11];
    result[14]=matrix1[2]*matrix2[12]+
      matrix1[6]*matrix2[13]+
      matrix1[10]*matrix2[14]+
      matrix1[14]*matrix2[15];
    result[3]=matrix1[3]*matrix2[0]+
      matrix1[7]*matrix2[1]+
      matrix1[11]*matrix2[2]+
      matrix1[15]*matrix2[3];
    result[7]=matrix1[3]*matrix2[4]+
      matrix1[7]*matrix2[5]+
      matrix1[11]*matrix2[6]+
      matrix1[15]*matrix2[7];
    result[11]=matrix1[3]*matrix2[8]+
```

```
    matrix1[7]*matrix2[9]+
    matrix1[11]*matrix2[10]+
    matrix1[15]*matrix2[11];
  result[15]=matrix1[3]*matrix2[12]+
    matrix1[7]*matrix2[13]+
    matrix1[11]*matrix2[14]+
    matrix1[15]*matrix2[15];
}


#define SWAP_ROWS(a, b) { double *_tmp = a; (a)=(b); (b)=_tmp; }
#define MAT(m,r,c) (m)[(c)*4+(r)]

int InvertMatrix(double *m, double *out){
  double wtmp[4][8];
  double m0, m1, m2, m3, s;
  double *r0, *r1, *r2, *r3;
  r0 = wtmp[0], r1 = wtmp[1], r2 = wtmp[2], r3 = wtmp[3];
  r0[0] = MAT(m, 0, 0), r0[1] = MAT(m, 0, 1),
    r0[2] = MAT(m, 0, 2), r0[3] = MAT(m, 0, 3),
    r0[4] = 1.0, r0[5] = r0[6] = r0[7] = 0.0,
    r1[0] = MAT(m, 1, 0), r1[1] = MAT(m, 1, 1),
    r1[2] = MAT(m, 1, 2), r1[3] = MAT(m, 1, 3),
    r1[5] = 1.0, r1[4] = r1[6] = r1[7] = 0.0,
    r2[0] = MAT(m, 2, 0), r2[1] = MAT(m, 2, 1),
    r2[2] = MAT(m, 2, 2), r2[3] = MAT(m, 2, 3),
    r2[6] = 1.0, r2[4] = r2[5] = r2[7] = 0.0,
    r3[0] = MAT(m, 3, 0), r3[1] = MAT(m, 3, 1),
    r3[2] = MAT(m, 3, 2), r3[3] = MAT(m, 3, 3),
    r3[7] = 1.0, r3[4] = r3[5] = r3[6] = 0.0;
  /* choose pivot - or die */
  if (fabs(r3[0]) > fabs(r2[0]))
    SWAP_ROWS(r3, r2);
  if (fabs(r2[0]) > fabs(r1[0]))
    SWAP_ROWS(r2, r1);
  if (fabs(r1[0]) > fabs(r0[0]))
    SWAP_ROWS(r1, r0);
  if (0.0 == r0[0])
    return 0;
  /* eliminate first variable     */
  m1 = r1[0] / r0[0];
  m2 = r2[0] / r0[0];
  m3 = r3[0] / r0[0];
  s = r0[1];
  r1[1] -= m1 * s;
```

```
    r2[1] -= m2 * s;
    r3[1] -= m3 * s;
    s = r0[2];
    r1[2] -= m1 * s;
    r2[2] -= m2 * s;
    r3[2] -= m3 * s;
    s = r0[3];
    r1[3] -= m1 * s;
    r2[3] -= m2 * s;
    r3[3] -= m3 * s;
    s = r0[4];
    if (s != 0.0) {
      r1[4] -= m1 * s;
      r2[4] -= m2 * s;
      r3[4] -= m3 * s;
    }
    s = r0[5];
    if (s != 0.0) {
      r1[5] -= m1 * s;
      r2[5] -= m2 * s;
      r3[5] -= m3 * s;
    }
    s = r0[6];
    if (s != 0.0) {
      r1[6] -= m1 * s;
      r2[6] -= m2 * s;
      r3[6] -= m3 * s;
    }
    s = r0[7];
    if (s != 0.0) {
      r1[7] -= m1 * s;
      r2[7] -= m2 * s;
      r3[7] -= m3 * s;
    }
    /* choose pivot - or die */
    if (fabs(r3[1]) > fabs(r2[1]))
      SWAP_ROWS(r3, r2);
    if (fabs(r2[1]) > fabs(r1[1]))
      SWAP_ROWS(r2, r1);
    if (0.0 == r1[1])
      return 0;
    /* eliminate second variable */
    m2 = r2[1] / r1[1];
    m3 = r3[1] / r1[1];
    r2[2] -= m2 * r1[2];
```

```
r3[2] -= m3 * r1[2];
r2[3] -= m2 * r1[3];
r3[3] -= m3 * r1[3];
s = r1[4];
if (0.0 != s) {
  r2[4] -= m2 * s;
  r3[4] -= m3 * s;
}
s = r1[5];
if (0.0 != s) {
  r2[5] -= m2 * s;
  r3[5] -= m3 * s;
}
s = r1[6];
if (0.0 != s) {
  r2[6] -= m2 * s;
  r3[6] -= m3 * s;
}
s = r1[7];
if (0.0 != s) {
  r2[7] -= m2 * s;
  r3[7] -= m3 * s;
}
/* choose pivot - or die */
if (fabs(r3[2]) > fabs(r2[2]))
  SWAP_ROWS(r3, r2);
if (0.0 == r2[2])
  return 0;
/* eliminate third variable */
m3 = r3[2] / r2[2];
r3[3] -= m3 * r2[3], r3[4] -= m3 * r2[4],
  r3[5] -= m3 * r2[5], r3[6] -= m3 * r2[6], r3[7] -= m3 * r2[7];
/* last check */
if (0.0 == r3[3])
  return 0;
s = 1.0 / r3[3];      /* now back substitute row 3 */
r3[4] *= s;
r3[5] *= s;
r3[6] *= s;
r3[7] *= s;
m2 = r2[3];           /* now back substitute row 2 */
s = 1.0 / r2[2];
r2[4] = s * (r2[4] - r3[4] * m2), r2[5] = s * (r2[5] - r3[5] * m2),
  r2[6] = s * (r2[6] - r3[6] * m2), r2[7] = s * (r2[7] - r3[7] * m2);
m1 = r1[3];
```

```
  r1[4] -= r3[4] * m1, r1[5] -= r3[5] * m1,
    r1[6] -= r3[6] * m1, r1[7] -= r3[7] * m1;
  m0 = r0[3];
  r0[4] -= r3[4] * m0, r0[5] -= r3[5] * m0,
    r0[6] -= r3[6] * m0, r0[7] -= r3[7] * m0;
  m1 = r1[2];             /* now back substitute row 1 */
  s = 1.0 / r1[1];
  r1[4] = s * (r1[4] - r2[4] * m1), r1[5] = s * (r1[5] - r2[5] * m1),
    r1[6] = s * (r1[6] - r2[6] * m1), r1[7] = s * (r1[7] - r2[7] * m1);
  m0 = r0[2];
  r0[4] -= r2[4] * m0, r0[5] -= r2[5] * m0,
    r0[6] -= r2[6] * m0, r0[7] -= r2[7] * m0;
  m0 = r0[1];             /* now back substitute row 0 */
  s = 1.0 / r0[0];
  r0[4] = s * (r0[4] - r1[4] * m0), r0[5] = s * (r0[5] - r1[5] * m0),
    r0[6] = s * (r0[6] - r1[6] * m0), r0[7] = s * (r0[7] - r1[7] * m0);
  MAT(out, 0, 0) = r0[4];
  MAT(out, 0, 1) = r0[5], MAT(out, 0, 2) = r0[6];
  MAT(out, 0, 3) = r0[7], MAT(out, 1, 0) = r1[4];
  MAT(out, 1, 1) = r1[5], MAT(out, 1, 2) = r1[6];
  MAT(out, 1, 3) = r1[7], MAT(out, 2, 0) = r2[4];
  MAT(out, 2, 1) = r2[5], MAT(out, 2, 2) = r2[6];
  MAT(out, 2, 3) = r2[7], MAT(out, 3, 0) = r3[4];
  MAT(out, 3, 1) = r3[5], MAT(out, 3, 2) = r3[6];
  MAT(out, 3, 3) = r3[7];
  return 1;
}

void Translate(double *result, double x, double y, double z){
  double matrix[16], resultMatrix[16];

  LoadIdentity(matrix);
  matrix[12] = x;
  matrix[13] = y;
  matrix[14] = z;

  MultiplyMatrices(resultMatrix, result, matrix);
  memcpy(result, resultMatrix, 16*sizeof(double));
}
```

3d.h

```c
/*
   This file is part of the Mandelbox program developed for the course
```

```
    CS/SE  Distributed Computer Systems taught by N. Nedialkov in the
    Winter of 2015-2016 at McMaster University.

    Copyright (C) 2015-2016 T. Gwosdz and N. Nedialkov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef _3d_H
#define _3d_H


#define NEAR 1
#define FAR 100

#include "camera.h"
#include "renderer.h"

void init3D(CameraParams *camP, const RenderParams *renP);

void   LoadIdentity   (double *matrix);
void   Perspective    (double fov, double aspect, double zNear, double zFar, double
*projMatrix);
void   Frustum        (double left, double right, double bottom, double top, double
znear, double zfar, double *matrix);
void   LookAt         (double *eye, double *target, double *up, double *modelMatrix)
;
double LengthVector   (double *vector);
void   NormalizeVector(double *vector);
void   ComputeNormalOfPlane(double *normal, double *v1, double *v2);
void   MultiplyMatrices(double *result, const double *matrix1, const double *matrix2
);
int    InvertMatrix(double *m, double *out);
void   Translate(double *result, double x, double y, double z);
```

```
#endif
```

camera.h

```c
/*
   This file is part of the Mandelbox program developed for the course
    CS/SE  Distributed Computer Systems taught by N. Nedialkov in the
    Winter of 2015-2016 at McMaster University.

    Copyright (C) 2015-2016 T. Gwosdz and N. Nedialkov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#ifndef _CAMERA_H
#define _CAMERA_H

#ifdef _OPENACC
#include <openacc.h>
#endif


typedef struct
{
  double camPos[3];
  double camTarget[3];
  double camUp[3];
  double fov;
  double matModelView[16];
  double matProjection[16];
  double matInvProjModel[16];
  int    viewport[4];
} CameraParams;

#endif
```

color.h

```c
/*
   This file is part of the Mandelbox program developed for the course
    CS/SE  Distributed Computer Systems taught by N. Nedialkov in the
    Winter of 2015-2016 at McMaster University.

    Copyright (C) 2015-2016 T. Gwosdz and N. Nedialkov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#ifndef COLOR_H
#define COLOR_H

#include "vector3d.h"

#ifdef _OPENACC
#include <openacc.h>
#endif


typedef struct
{
  bool escaped;
  vec3 hit;
  vec3 normal;
} pixelData;

#endif
```

genvideo.sh

```bash
#!/bin/bash
# script called if -v flag is present. Stitches together all frames after they have been rendered
cd ../frames;
echo -n "Generating video. This may take a while...  "
rename 's/\d+/sprintf("%05d",$&)/e' *.bmp
ffmpeg -y -framerate 30 -i %05d.bmp -c:v libx264 ../out.mp4 &> /dev/null
echo "done."
```

getcolor.cc

```c
/*
   This file is part of the Mandelbox program developed for the course
    CS/SE  Distributed Computer Systems taught by N. Nedialkov in the
    Winter of 2015-2016 at McMaster University.

    Copyright (C) 2015-2016 T. Gwosdz and N. Nedialkov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#include "color.h"
#include "renderer.h"
#include "vector3d.h"
#include <cmath>
#include <algorithm>

#ifdef _OPENACC
#include <openacc.h>
#include <accelmath.h>
#endif

// CHANGES FOR OPENACC
```

```
// lighting is now inlined to avoid a calling depth greater than one
// getcolor takes explicit renderer parameters rather than passing a structure


// 3 new color schemes can be specified in the params file.


inline void lighting(const vec3 &n, const vec3 &color, const vec3 &pos, const vec3 &
direction,  vec3 &outV)
{
  vec3 CamLight = { 1.0, 1.0, 1.0};
  double CamLightW = 1.8;// 1.27536;
  double CamLightMin = 0.3;// 0.48193;

  vec3 nn = { n.x -1.0, n.y -1, n.z -1 };
  double dot_res = nn.x * direction.x + nn.y * direction.y + nn.z * direction.z;
  double ambient = MAX( CamLightMin, dot_res ) * CamLightW;

  outV.x = CamLight.x * ambient * color.x;
  outV.x = CamLight.y * ambient * color.y;
  outV.x = CamLight.z * ambient * color.z;

}

#pragma acc routine seq
void getcolor(const pixelData &pixData, const int colorType, const float brightness,
 //const RenderParams render_params,
           const vec3 &from, const vec3  &direction, vec3 &result)
{

  /* COLOR SCHEMES

      0, 1: default
      2   : red filter
      3   : grayscale
      4   : less flamboyant rainbow
  */


  vec3 baseColor = {1.0, 1.0, 1.0};
  vec3 backColor = {0.4, 0.4, 0.4};

  //coloring and lightning
  vec3 hitColor =  {baseColor.x, baseColor.y, baseColor.z};
```

```
if (pixData.escaped == false)
{
    //apply lighting
    lighting(pixData.normal, hitColor, pixData.hit, direction, hitColor);

    //add normal based coloring
    if(0 <= colorType <= 4)
      {

      hitColor.x = (hitColor.x * pixData.normal.x + 1.0)/2.0 * brightness;
      hitColor.y = (hitColor.y * pixData.normal.y + 1.0)/2.0 * brightness;
      hitColor.z = (hitColor.z * pixData.normal.z + 1.0)/2.0 * brightness;

        //gamma correction
        v_clamp(hitColor, 0.0, 1.0);
      SQUARE(hitColor);

      }
    if(colorType == 1)
      {
        //"swap" colors
        double t = hitColor.x;
        hitColor.x = hitColor.z;
        hitColor.z = t;
      } else if (colorType == 2)
    {
      // red filter
      hitColor.x = 0.0;
    } else if (colorType == 3)
    {
      // grayscale
      //weighted average used by GIMP based on human perception
      //double avg = 0.21 * hitColor.x + 0.72 * hitColor.y + 0.07 * hitColor.z;
      double avg = (hitColor.x + hitColor.y + hitColor.z) / 3.0;
      hitColor.x = avg;
      hitColor.y = avg;
      hitColor.z = avg;
    } else
    {
      // rainbow
      hitColor.x = 0.85 * hitColor.x;
      hitColor.z = 0.7 * hitColor.z;
    }

}
```

```
  else {
    //we have the background color
    hitColor = backColor;
  }

  result.x = hitColor.x;
  result.y = hitColor.y;
  result.z = hitColor.z;

}
```

getparams.cc

```c
/*
   This file is part of the Mandelbox program developed for the course
    CS/SE  Distributed Computer Systems taught by N. Nedialkov in the
    Winter of 2015-2016 at McMaster University.

    Copyright (C) 2015-2016 T. Gwosdz and N. Nedialkov
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "renderer.h"
#include "mandelbulb.h"
#include "mandelbox.h"
#include "camera.h"

#define BUF_SIZE 1024

static char buf[BUF_SIZE];


// CHANGES FOR OPENACC
// Parameters are now explicitly different for MandelBox and MandelBulb. The .dat fi
les themselves
//    are still interchangeable
// i.e., rMin and rMax are named escape_time and power for a bulb
// The appropriate parameter struct is populated depending on compiler flag -DBULB o
r -DBOX


#ifdef BULB
void getParameters(char *filename, CameraParams *camP, RenderParams *renP, MandelBul
```

```
bParams *bulbP)
#else
void getParameters(char *filename, CameraParams *camP, RenderParams *renP, MandelBox
Params *boxP)
#endif
{
  FILE *fp;
  int ret;
  double *d;

  renP->fractalType = 0;
  renP->maxRaySteps = 8000;
  renP->maxDistance = 1000;

  fp = fopen(filename,"r");

  if( !fp )
    {
      printf(" *** File %s does not exist\n", filename);
      exit(1);
    }

  int count = 0;

  while (1)
    {
      memset(buf, 0, BUF_SIZE);

      ret = fscanf(fp, "%1023[^\n]\n", buf);
      if (ret == EOF) break;

      if(buf[0] == '#') // comment line
    continue;

      switch(count)
    {
      // CAMERA
      //camera position
    case 0:
      d = camP->camPos;
      sscanf(buf, "%lf %lf %lf", d, d+1, d+2);
      break;
    case 1:
      //camera target
      d = camP->camTarget;
```

```
      sscanf(buf, "%lf %lf %lf", d, d+1, d+2);
      break;
      //camera up
    case 2:
      d = camP->camUp;
      sscanf(buf, "%lf %lf %lf", d, d+1, d+2);
      break;
      //field of view
    case 3:
      sscanf(buf, "%lf", &camP->fov);
      break;

      //IMAGE
      //width, height
    case 4:
      sscanf(buf, "%d %d", &renP->width, &renP->height);
      break;
      //detail
    case 5:
      sscanf(buf, "%f", &renP->detail);
      break;

      //FRACTAL
    case 6:
    // box: scale, rmin, rfixed
    // bulb: IGNORE, escape time(bailout), power
      //sscanf(buf, "%f %f %f", &boxP->scale, &boxP->rMin, &boxP->rFixed);
  #ifdef BULB
    sscanf(buf, "%*f %f %f", &bulbP->escape_time, &bulbP->power);
  #else
    sscanf(buf, "%f %f %f", &boxP->scale, &boxP->rMin, &boxP->rFixed);
  #endif
    break;

    case 7:
      //sscanf(buf, "%d %f ", &boxP->num_iter, &boxP->escape_time);
    // bulb: max iterations, IGNORE
  #ifdef BULB
    sscanf(buf, "%d %*f ", &bulbP->num_iter);
  #else
    sscanf(buf, "%d %f ", &boxP->num_iter, &boxP->escape_time);
  #endif
    break;

      //COLORING
```

```
    case 8:
      sscanf(buf, "%d", &renP->colorType);
      break;
    case 9:
    // brightness
      sscanf(buf, "%f ", &renP->brightness);
      break;
      //FILENAME
    case 10:
      strcpy(renP->file_name, buf);
      break;
    }
      count++;
    }
  fclose(fp);
}
```

init3D.cc

```c
/*
   This file is part of the Mandelbox program developed for the course
    CS/SE  Distributed Computer Systems taught by N. Nedialkov in the
    Winter of 2015-2016 at McMaster University.

    Copyright (C) 2015-2016 T. Gwosdz and N. Nedialkov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "camera.h"
#include "renderer.h"
#include "3d.h"
```

```
//UNCHANGED FOR OPENACC

void init3D(CameraParams *camP, const RenderParams *renP)
{
  //set up the viewport for the image
  camP->viewport[0] = 0;
  camP->viewport[1] = 0;
  camP->viewport[2] = renP->width;
  camP->viewport[3] = renP->height;

  //init the matricies
  LoadIdentity(camP->matModelView);
  LoadIdentity(camP->matProjection);

  //setting up camera lense
  Perspective((65*camP->fov), ((double)renP->width)/((double)renP->height), NEAR, FA
R, camP->matProjection);

  //setting up model view matrix
  LookAt(camP->camPos, camP->camTarget, camP->camUp, camP->matModelView);

  //setting up the inverse(projection x model) matrix
  double temp[16];
  MultiplyMatrices(temp, camP->matProjection, camP->matModelView);
  //Now compute the inverse of matrix A
  InvertMatrix(temp, camP->matInvProjModel);
}
```

main.cc

```c
/*
   This file is part of the Mandelbox program developed for the course
    CS/SE  Distributed Computer Systems taught by N. Nedialkov in the
    Winter of 2015-2016 at McMaster University.

    Copyright (C) 2015-2016 T. Gwosdz and N. Nedialkov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
```

```
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "camera.h"
#include "renderer.h"
#include "mandelbulb.h"
#include "mandelbox.h"
#include "walk.h"
#include <unistd.h>


//Compiler flags -DBULB or -DBOX

#ifdef BULB

void getParameters(char *filename, CameraParams *camera_params, RenderParams *render
er_params,
           MandelBulbParams *mandelBulb_paramsP);
void renderFractal(const CameraParams camera_params, const RenderParams renderer_par
ams,
                  const MandelBulbParams bulb_params, unsigned char* image, int fram
e);
MandelBulbParams mandelBulb_params;

#else

void getParameters(char *filename, CameraParams *camera_params, RenderParams *render
er_params,
       MandelBoxParams *mandelBox_paramsP);
void renderFractal(const CameraParams camera_params, const RenderParams renderer_par
ams,
                  const MandelBoxParams box_params, unsigned char* image, int frame)
;
MandelBoxParams mandelBox_params;

#endif

void init3D       (CameraParams *camera_params, const RenderParams *renderer_params)
```

```
;
void saveBMP      (const char* filename, const unsigned char* image, int width, int
height);

int main(int argc, char** argv)
{
  // Get parameters:
  int vflag = 0,
      verbose = 1,
      num_of_iterations = 1,
      start_frame = 0,
      c;
  opterr = 0;
  char * fname = argv[1];

  while ((c = getopt(argc, argv, "hvnf:")) != -1)
    switch (c)
    {
      case 'h':
        printf("Usage:");
        #ifdef BULB
          printf(" ./mandelbox ");
        #else
          printf(" ./mandelbulb ");
        #endif
        printf("-hvnfs\n");
        printf("   -h   : Display this help message\n");
        printf("   -v   : Generate video from frames\n");
        printf("   -n   : Less verbose output\n");
        printf("   -f n : Generate n frames\n");
        return 0;
      case 'v':
        vflag = 1;
        break;
      case 'n':
        verbose = 0;
        break;
      case 'f':
        if (optarg == NULL){
          printf("Invalid option -f\nInteger required. Ex: -n 100\n");
          return 1;
        }
        num_of_iterations = atoi(optarg);
        break;
      default:
```

```
        printf("Unknwon Option. Aborting.\n");
        return 1;
  }

  int frame, i;
  RenderParams renderer_params;
  CameraParams camera_history [num_of_iterations + 1];

  // Get bulb/box params
  for (i=0;i<num_of_iterations;i++){
    #ifdef BULB
      getParameters(fname, &camera_history[i], &renderer_params, &mandelBulb_params)
;
    #else
      getParameters(fname, &camera_history[i], &renderer_params, &mandelBox_params);
    #endif
    init3D(&camera_history[i], &renderer_params);
  }
    // Initialize params and image
  int image_size = renderer_params.width * renderer_params.height;
  unsigned char *image = (unsigned char*)malloc(3*image_size*sizeof(unsigned char));

  // Verbose output. Silence with -n
  if (verbose) {
    printf("Image Size:       %dx%d\n", renderer_params.width, renderer_params.heigh
t);
    printf("Video:            %s\n", vflag ? "Yes" : "No");
      printf("Number of Frames: %d", num_of_iterations);
    if (num_of_iterations == 1) {
      printf(" (for more frames, use -f)\n\n");
    } else { printf("\n\n"); }
  }

  for (frame = start_frame; frame < num_of_iterations; frame++){

    // Generate unique image name
    char buf[15];
    sprintf(buf, "../frames/%05d.bmp", frame);

    if (verbose) {
      printf("Rendering frame: %d\n", frame);
    }

    // Compute the next camera position and render the frame
    // All Parallelization is in renderFractal
```

```
    #ifdef BULB
      // Mandelbulb
      walk(camera_history, &renderer_params, &mandelBulb_params, verbose, frame);
      renderFractal(camera_history[frame], renderer_params, mandelBulb_params, image
, frame);
    #else
      // Mandelbox
      walk(camera_history, &renderer_params, &mandelBox_params, verbose, frame);
      renderFractal(camera_history[frame], renderer_params, mandelBox_params, image,
 frame);
    #endif

    // Save image
    saveBMP(buf, image, renderer_params.width, renderer_params.height);
  }

  // Cleanup
  free(image);

  // Video shell script
  if (vflag){
    system("./genvideo.sh");
  }

  return 0;
}
```

mandelbox.h

```c
/*
   This file is part of the Mandelbox program developed for the course
    CS/SE  Distributed Computer Systems taught by N. Nedialkov in the
    Winter of 2015-2016 at McMaster University.

    Copyright (C) 2015-2016 T. Gwosdz and N. Nedialkov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#ifndef MANDELBOX_H
#define MANDELBOX_H


typedef struct {
  float rMin, rFixed;
  float scale;
  float escape_time;
  int num_iter;
} MandelBoxParams;

#endif
```

mandelbulb.h

```c
/*
   This file is part of the Mandelbox program developed for the course
    CS/SE  Distributed Computer Systems taught by N. Nedialkov in the
    Winter of 2015-2016 at McMaster University.

    Copyright (C) 2015-2016 T. Gwosdz and N. Nedialkov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#ifndef MANDELBULB_H
#define MANDELBULB_H

#ifdef _OPENACC
#include <openacc.h>
#endif

// Distinct from box parameters

typedef struct {
  float escape_time;
  float power;
  int num_iter;
} MandelBulbParams;

#endif
```

print.cc

```c
/*
   This file is part of the Mandelbox program developed for the course
    CS/SE  Distributed Computer Systems taught by N. Nedialkov in the
    Winter of 2015-2016 at McMaster University.

    Copyright (C) 2015-2016 T. Gwosdz and N. Nedialkov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#include <stdio.h>
#include <string.h>
#include <math.h>

void printProgress( double perc, double time , int frame)
{
  static char delete_space[80];
  static char * OutputString;
  perc *= 100;

  int sec = ceil(time);
  int hr = sec/3600;
  int t = sec%3600;
  int min = t/60;
  sec = t%60;

  OutputString = (char*)"*** completed % 5.2f%s of frame %d ---  total time = %02d:%02d:%02d   % e (s)";
  sprintf(delete_space, OutputString, perc, "%%", frame, hr, min, sec, time);

  fprintf( stderr, delete_space);
  for ( unsigned int i = 0; i < strlen(delete_space); i++)
    fputc( 8, stderr);
}
```

raymarching.cc

```c
/*
   This file is part of the Mandelbox program developed for the course
    CS/SE  Distributed Computer Systems taught by N. Nedialkov in the
    Winter of 2015-2016 at McMaster University.

    Copyright (C) 2015-2016 T. Gwosdz and N. Nedialkov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#include <assert.h>
#include <algorithm>
#include <stdio.h>

#include "color.h"
#include "renderer.h"
#include "mandelbulb.h"
#include "mandelbox.h"

#ifdef _OPENACC
#include <openacc.h>
#include <accelmath.h>
#else
#include <math.h>
#endif

//CHANGES FOR OPENACC
//DE is now inlined to avoid a calling depth greater than one
//Seperate functions for bulb and box


#ifdef BULB //bulb DE
```

```
inline double DE(const vec3 &p0,
  const float escape_time, const float power, const int num_iter)
{
  vec3 z = p0;

  double dr = 1.0;
  double r = 0.0;

  double Bailout = escape_time;
  double Power = power;

  for (int i=0; i < num_iter; i++)
    {
      MAGNITUDE(r,z);
      if(r > Bailout) break;

      double theta = acos(z.z/r);
      double phi   = atan2(z.y, z.x);
      dr = pow(r, Power - 1.0) * Power * dr + 1.0;

      double zr = pow(r, Power);
      theta     = theta * Power;
      phi       = phi * Power;

      z.x = zr*sin(theta)*cos(phi);
      z.y = zr*sin(phi)*sin(theta);
      z.z = zr*cos(theta);

      z.x = z.x + p0.x;
      z.y = z.y + p0.y;
      z.z = z.z + p0.z;
    }

  return 0.5*log(r)*r/dr;
}

#else // BOX DE + macros, copysign function

// duplication of math lib copysign unavailable in OpenACC
inline double copysign(double x, double y){

  if(y < -0.00000000000001){
    return -fabs(x);
  }else{
```

```
    return fabs(x);
  }
}

#define SQR(x) ((x)*(x))
#define COMPONENT_FOLD(x) { (x) = fabs(x) <= 1? (x) : copysign(2,(x))-(x); }

inline double DE(const vec3 &p0, const int num_iter, const float rMin,
  const float rFixed, const float escape_time, const float scale, double c1, double
c2)
{

  vec3 p = p0;
  double rMin2   = SQR(rMin);
  double rFixed2 = SQR(rFixed);
  double escape  = SQR(escape_time);
  double dfactor = 1;
  double r2      =-1;
  const double rFixed2rMin2 = rFixed2/rMin2;

  int i = 0;
  while (i< num_iter && r2 < escape)
    {
      COMPONENT_FOLD(p.x);
      COMPONENT_FOLD(p.y);
      COMPONENT_FOLD(p.z);

      DOT(r2,p);

      if (r2<rMin2)
  {
    MULTIPLY_BY_DOUBLE(p, rFixed2rMin2);
    dfactor *= rFixed2rMin2;
  }
      else
      if ( r2<rFixed2)
  {
    const double t = (rFixed2/r2);
    MULTIPLY_BY_DOUBLE(p, t);
    dfactor *= t;
  }


      dfactor = dfactor*fabs(scale)+1.0;
      p.x = p.x * scale + p0.x;
```

```
      p.y = p.y * scale + p0.y;
      p.z = p.z * scale + p0.z;

      i++;
    }

  double r = 0.0;
  MAGNITUDE(r, p);
  r -= c1;
  r = r / dfactor;
  r -= c2;

  return  r;
}

#endif


// RAYMARCH
// all renderer, camera, and box/bulb params are passed explicitly

#ifdef BULB
#pragma acc routine seq
double rayMarch(const int maxRaySteps, const float maxDistance,
 const float escape_time, const float power, const int num_iter,
 const vec3 &from, const vec3  &direction, double eps, pixelData& pix_data)
#else
#pragma acc routine seq
double rayMarch(const int maxRaySteps, const float maxDistance,
 const int num_iter, const float rMin, const float rFixed, const float escape_time,
const float scale,
 const vec3 &from, const vec3  &direction, double eps, pixelData& pix_data)
#endif
{

  double dist = 0.0;
  double totalDist = 0.0;

  #ifdef BOX
  double c1 = fabs(scale - 1.0);
  double c2 = pow( fabs(scale), 1 - num_iter);
  #endif

  const double sqrt_mach_eps = 1.4901e-08;
```

```
// We will adjust the minimum distance based on the current zoom

double epsModified = 0.0;

int steps=0;
vec3 p;
do
  {
    //p = from + direction * totalDist;
    VEC(p,
      from.x + direction.x * totalDist,
      from.y + direction.y * totalDist,
      from.z + direction.z * totalDist
      );

    //dist = DE(p, bulb_params);
    #ifdef BULB
    dist = DE(p, escape_time, power, num_iter);
    #else
    dist = DE(p, num_iter, rMin,
      rFixed, escape_time, scale, c1, c2);
    #endif


    totalDist += .95*dist;

    epsModified = totalDist;
    epsModified*=eps;
    steps++;
  }
while (dist > epsModified && totalDist <= maxDistance && steps < maxRaySteps);

//vec3 hitNormal; unused

if (dist < epsModified)
  {
    //we didnt escape
    pix_data.escaped = false;

    // We hit something, or reached MaxRaySteps
    pix_data.hit = p;

    //figure out the normal of the surface at this point
    //const vec3 normPos = p - direction * epsModified;
    const vec3 normPos = {
```

```
    p.x - direction.x * epsModified,
    p.y - direction.y * epsModified,
    p.z - direction.z * epsModified
  };


  // compute the normal at p
  double eps;
  MAGNITUDE(eps, normPos) ;
  eps = MAX(eps, 1.0);
  eps *= sqrt_mach_eps;

  // precompute the vectors passed to DE
  vec3 e1 = {eps, 0, 0};
  vec3 e2 = {0, eps, 0};
  vec3 e3 = {0, 0, eps};
  vec3 vs1, vs2, vs3;
  vec3 vd1, vd2, vd3;
  VECTOR_SUM(vs1, normPos,e1);
  VECTOR_SUM(vs2, normPos,e2);
  VECTOR_SUM(vs3, normPos,e3);
  VECTOR_DIFF(vd1, normPos, e1);
  VECTOR_DIFF(vd2, normPos, e2);
  VECTOR_DIFF(vd3, normPos, e3);

  #ifdef BULB
  pix_data.normal.x = DE(vs1, escape_time, power, num_iter)-DE(vd1, escape_time,
power, num_iter);
  pix_data.normal.y = DE(vs2, escape_time, power, num_iter)-DE(vd2, escape_time,
power, num_iter);
  pix_data.normal.z = DE(vs3, escape_time, power, num_iter)-DE(vd3, escape_time,
power, num_iter);
  #else
  pix_data.normal.x =
    DE(vs1, num_iter, rMin,rFixed, escape_time, scale, c1, c2)
    -DE(vd1, num_iter, rMin, rFixed, escape_time, scale, c1, c2);
  pix_data.normal.y =
    DE(vs2, num_iter, rMin, rFixed, escape_time, scale, c1, c2)
    -DE(vd2, num_iter, rMin, rFixed, escape_time, scale, c1, c2);
  pix_data.normal.z =
    DE(vs3, num_iter, rMin, rFixed, escape_time, scale, c1, c2)
    -DE(vd3, num_iter, rMin, rFixed, escape_time, scale, c1, c2);
  #endif

  NORMALIZE(pix_data.normal);
```

```
    }
  else {
    //we have the background color
    pix_data.escaped = true;
    return 0;
  }

  return dist;
}
```

renderer.cc

```c
/*
   This file is part of the Mandelbox program developed for the course
    CS/SE  Distributed Computer Systems taught by N. Nedialkov in the
    Winter of 2015-2016 at McMaster University.

    Copyright (C) 2015-2016 T. Gwosdz and N. Nedialkov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#include <stdio.h>
#include <stdlib.h>

#include "color.h"
#include "mandelbulb.h"
#include "mandelbox.h"
#include "camera.h"
#include "vector3d.h"
#include "3d.h"

#ifdef _OPENACC
#include <openacc.h>
```

```
#endif

// All parallelization is done in renderFractal
// raymarch, getcolor, and unproject are sequential openacc routines
// vec3 has been converted from class to struct with macros (see vector3d.h)
// The cost for distinct Mandelbulb and Mandelbox params is duplicate method signatu
res
// Specified with -DBULB or -DBOX compiler flag

extern double getTime();
extern void   printProgress( double perc, double time, int frame );

#ifdef BULB
#pragma acc routine seq
extern double rayMarch(const int maxRaySteps, const float maxDistance,
  const float escape_time, const float power, const int num_iter,
  const vec3 &from, const vec3 &direction, double eps, pixelData& pix_data);
#else
#pragma acc routine seq
extern double rayMarch(const int maxRaySteps, const float maxDistance,
  const int num_iter, const float rMin, const float rFixed, const float escape_time,
 const float scale,
  const vec3 &from, const vec3 &direction, double eps, pixelData& pix_data);
#endif

#pragma acc routine seq
extern void getcolor(const pixelData &pixData, const int colorType, const float brig
htness,
            const vec3 &from, const vec3  &direction, vec3 &result);

#pragma acc routine seq
extern void UnProject(double winX, double winY, const int viewport[4], const double
matInvProjModel[16], vec3 &obj);


#ifdef BULB
void renderFractal(const CameraParams camera_params, const RenderParams renderer_par
ams,
                   const MandelBulbParams bulb_params, unsigned char* image, int fr
ame)
#else
void renderFractal(const CameraParams camera_params, const RenderParams renderer_par
ams,
                   const MandelBoxParams box_params, unsigned char* image, int fram
e)
```

```
#endif
{
  // DIRECTION, COLOR, PIXEL ARRAYS
  // OpenACC has problems with incorrectly sharing structs
  // Solved by creating struct array (one per pixel, or loop iteration) that is shar
ed in parallel region
  int size = renderer_params.width * renderer_params.height;
#ifdef _OPENACC
    vec3* direction = (vec3*)acc_malloc(size * sizeof(vec3));
    pixelData* pixel = (pixelData*)acc_malloc(size * sizeof(pixelData));
    vec3* color = (vec3*)acc_malloc(size * sizeof(vec3));
#else
    vec3* direction = (vec3*)malloc(size * sizeof(vec3));
    pixelData* pixel = (pixelData*)malloc(size * sizeof(pixelData));
    vec3* color = (vec3*)malloc(size * sizeof(vec3));
#endif

  // All parameters are explicitly copied into ACC device region
  // parameter structs are no longer passed as function arguments

  // RENDERER PARAMS
  const int colorType = renderer_params.colorType;
  const float brightness = renderer_params.brightness;
  const int height = renderer_params.height;
  const int width  = renderer_params.width;
  const float detail = renderer_params.detail;
  const int maxRaySteps = renderer_params.maxRaySteps;
  const float maxDistance = renderer_params.maxDistance;

  // CAMERA PARAMS
  const double camPos[3] = {camera_params.camPos[0], camera_params.camPos[1], camera
_params.camPos[2]} ;
  const double matInvProjModel[16] =
  {
    camera_params.matInvProjModel[0],
    camera_params.matInvProjModel[1],
    camera_params.matInvProjModel[2],
    camera_params.matInvProjModel[3],
    camera_params.matInvProjModel[4],
    camera_params.matInvProjModel[5],
    camera_params.matInvProjModel[6],
    camera_params.matInvProjModel[7],
    camera_params.matInvProjModel[8],
    camera_params.matInvProjModel[9],
    camera_params.matInvProjModel[10],
```

```
    camera_params.matInvProjModel[11],
    camera_params.matInvProjModel[12],
    camera_params.matInvProjModel[13],
    camera_params.matInvProjModel[14],
    camera_params.matInvProjModel[15]
  };
  const int viewport[4] =
  {
    camera_params.viewport[0],
    camera_params.viewport[1],
    camera_params.viewport[2],
    camera_params.viewport[3]
  };

  printf("(%lf, %lf, %lf)\n",camera_params.camPos[0], camera_params.camPos[1], camer
a_params.camPos[2]);

  // copy bulb/box params into device region

  #ifdef BULB
    // MANDELBULB PARAMS
    const  float escape_time =   bulb_params.escape_time;
    const float power =  bulb_params.power;
    const int num_iter =  bulb_params.num_iter;

    #pragma acc enter data pcopyin(    \
      escape_time, \
      power, \
      num_iter \
    )
  #else

    // MANDELBOX PARAMS
    const int num_iter = box_params.num_iter;
    const float rMin = box_params.rMin;
    const float rFixed = box_params.rFixed;
    const float escape_time = box_params.escape_time;
    const float scale = box_params.scale;

    #pragma acc enter data pcopyin(    \
      rMin, \
      rFixed, \
      escape_time, \
      scale, \
      num_iter \
```

```
  )

#endif

// Copy in image, remaining parameters, and vec3 arrays
#pragma acc data present_or_copy(image[0:size*3]),   \
pcopyin(                  \
  camPos[:3],      \
  matInvProjModel[:16], \
  viewport[:4], \
                 \
  colorType, \
  brightness, \
  height, \
  width, \
  detail, \
  maxRaySteps, \
  maxDistance \
),             \
  deviceptr(direction, pixel, color)
{

// BEGIN DEVICE DATA REGION

#ifndef _OPENACC
double time = getTime();
#endif

const double eps = pow(10.0, detail);
const vec3 from = {camPos[0], camPos[1], camPos[2]};

// for some reason needed for compiler to parallelize loops
const int cheight = height;
const int cwidth = width;

int i,j;
// total of three parallel pragmas + external routine pragmas
#pragma acc parallel
#pragma acc loop
for(j = 0; j < cheight; j++)
{
  #pragma acc loop
  for(i = 0; i < cwidth; i++)
  {
```

```
      int k, l;
      //vec3 array index
      l = (j * width + i );

      // get point on the 'far' plane
      // acc routine
      UnProject(i, j, viewport, matInvProjModel, direction[l]);

      SUBTRACT_DOUBLE_ARRAY(direction[l], camPos);
      NORMALIZE( direction[l] );

      //render the pixel
      //acc routine
      //difference in box/bulb is the distance estimator used by raymarch
      #ifdef BULB
      rayMarch(maxRaySteps, maxDistance, escape_time, power, num_iter, from, directi
on[l], eps, pixel[l]);
      #else
      rayMarch(maxRaySteps, maxDistance, num_iter, rMin,
        rFixed, escape_time, scale, from, direction[l], eps, pixel[l]);
      #endif

      //get the color at this pixel
      getcolor(pixel[l], colorType, brightness, from, direction[l], color[l]);

      //save color into texture
      k = (j * width + i)*3;
      image[k+2] = (unsigned char)(color[l].x * 255);
      image[k+1] = (unsigned char)(color[l].y * 255);
      image[k]   = (unsigned char)(color[l].z * 255);

    }

      #ifndef _OPENACC
      printProgress((j+1)/(double)height,getTime()-time, frame);
      #endif
  }

  }// END DEVICE DATA REGION

  #ifdef _OPENACC
    // free memory used by acc for vec3 arrays
    acc_free(direction);
    acc_free(pixel);
    acc_free(color);
```

```
  #endif
}
```

renderer.h

```c
/*
   This file is part of the Mandelbox program developed for the course
    CS/SE  Distributed Computer Systems taught by N. Nedialkov in the
    Winter of 2015-2016 at McMaster University.

    Copyright (C) 2015-2016 T. Gwosdz and N. Nedialkov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef _RENMandelBulbDERER_H
#define _RENMandelBulbDERER_H

#ifdef _OPENACC
#include <openacc.h>
#endif

typedef struct
{
  int fractalType;
  int colorType;
  int super_sampling;
  float brightness;
  int width;
  int height;
  float detail;
  int maxRaySteps;
  float maxDistance;
  char file_name[80];
} RenderParams;

#endif
```

savebmp.cc

```c
/*
   This file is part of the Mandelbox program developed for the course
    CS/SE  Distributed Computer Systems taught by N. Nedialkov in the
    Winter of 2015-2016 at McMaster University.

    Copyright (C) 2015-2016 T. Gwosdz and N. Nedialkov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <math.h>

void saveBMP(const char* filename, const unsigned char* result, int w, int h){

    FILE *f;
    unsigned char *img = NULL;
    int filesize = 54 + 3*w*h;  //w is your image width, h is image height, both int

    unsigned char bmpfileheader[14] = {'B','M', 0,0,0,0, 0,0, 0,0, 54,0,0,0};
    unsigned char bmpinfoheader[40] = {40,0,0,0, 0,0,0,0, 0,0,0,0, 1,0, 24,0};
    unsigned char bmppad[3] = {0,0,0};

    bmpfileheader[ 2] = (unsigned char)(filesize    );
    bmpfileheader[ 3] = (unsigned char)(filesize>> 8);
    bmpfileheader[ 4] = (unsigned char)(filesize>>16);
    bmpfileheader[ 5] = (unsigned char)(filesize>>24);

    bmpinfoheader[ 4] = (unsigned char)(       w    );
    bmpinfoheader[ 5] = (unsigned char)(       w>> 8);
```

```
    bmpinfoheader[ 6] = (unsigned char)(       w>>16);
    bmpinfoheader[ 7] = (unsigned char)(       w>>24);
    bmpinfoheader[ 8] = (unsigned char)(       h    );
    bmpinfoheader[ 9] = (unsigned char)(       h>> 8);
    bmpinfoheader[10] = (unsigned char)(       h>>16);
    bmpinfoheader[11] = (unsigned char)(       h>>24);

    f = fopen(filename,"wb");
    fwrite(bmpfileheader,1,14,f);
    fwrite(bmpinfoheader,1,40,f);


    img = (unsigned char *)malloc(3*w);
    assert(img);

    int i,j;
    for(j=0; j<h; j++)
    {
        for(i=0; i<w; i++)
        {
            img[i*3+0] = result[(j*w+i)*3+0];
            img[i*3+1] = result[(j*w+i)*3+1];
            img[i*3+2] = result[(j*w+i)*3+2];
        }
        fwrite(img,3,w,f);
        fwrite(bmppad,1,(4-(w*3)%4)%4,f);
    }
    fclose(f);

}
```

timing.cc

```c
/*
   This file is part of the Mandelbox program developed for the course
    CS/SE  Distributed Computer Systems taught by N. Nedialkov in the
    Winter of 2015-2016 at McMaster University.

    Copyright (C) 2015-2016 T. Gwosdz and N. Nedialkov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#include <stdio.h>

#if defined(_MSC_VER) || defined(__MINGW32__)
#else
#include <sys/resource.h>
#endif
#include <unistd.h>

double getTime() {
#if defined(_MSC_VER) || defined(__MINGW32__)
  return 0;
#else
  struct rusage usage;
  getrusage(RUSAGE_SELF, &usage);
  struct timeval time;
  time = usage.ru_utime;
  return time.tv_sec+time.tv_usec/1e6;
#endif
}
```

vector3d.h

```c
#ifndef vec3_h
```

```
#define vec3_h

#ifdef _OPENACC
#include <accelmath.h>
#else
#include <math.h>
#endif

// vec3 has been changed to struct with accompanying macros
typedef struct
{
  double x, y, z;
}  vec3;

// set vec3 p = v
#define SET_POINT(p,v) { p.x=v.x; p.y=v.y; p.z=v.z; }
// set vec3 x, y, z to double[0..2]
#define SET_DOUBLE_POINT(p,v) { p.x=v[0]; p.y=v[1]; p.z=v[2]; }
// set vec3 x,y,z to v[i] - u[i]
#define SUBTRACT_POINT(p,v,u)           \
  {                        \
  p.x=(v[0])-(u[0]);                \
  p.y=(v[1])-(u[1]);                \
  p.z=(v[2])-(u[2]);                \
}
// x,y,z = x,y,z - double[0..2]
#define SUBTRACT_DOUBLE_ARRAY(v, d) {v.x = v.x - d[0]; v.y = v.y - d[1]; v.z = v.z -
 d[2];  }
// (x,y,z)^2
#define SQUARE(p)\
    {\
        p.x = p.x * p.x; \
        p.y = p.y * p.y; \
        p.z = p.z * p.z; \
    }
// normalize vector
#define NORMALIZE(p) {                  \
    double fMag = ( p.x*p.x + p.y*p.y + p.z*p.z );  \
    if (fMag != 0)                  \
      {                         \
    double fMult = 1.0/sqrt(fMag);          \
    p.x *= fMult;                   \
    p.y *= fMult;                   \
    p.z *= fMult;                   \
      }                         \
```

```
  }
// x*p, y*q, z*r
#define MULTIPLY_BY_VECTOR(v, p) ( { v.x = v.x*p.x; v.y = v.y*p.y; v.z = v.z*p.z; }
)
// x,y,z * d
#define MULTIPLY_BY_DOUBLE(v, d) ( { v.x = v.x*d; v.y = v.y*d; v.z = v.z*d; } )
// get vector magnitude
#define MAGNITUDE(m,p)  ({ m=sqrt( p.x*p.x + p.y*p.y + p.z*p.z ); })
// vector dot product
#define DOT(d,p) ({  d= p.x*p.x + p.y*p.y + p.z*p.z ; })
#define MAX(a,b) ( ((a)>(b))? (a):(b) )
// constructor
#define VEC(v,a,b,c) { v.x = a; v.y = b; v.z = c; }


inline double clamp(double d, double min, double max)
{
  const double t = d < min ? min : d;
  return t > max ? max : t;
}

// vector addition and subtraction
#define VECTOR_SUM(r, v1, v2) {  r.x = v1.x + v2.x; r.y = v1.y + v2.y; r.z = v1.z +
v2.z; }
#define VECTOR_DIFF(r, v1, v2) {  r.x = v1.x - v2.x; r.y = v1.y - v2.y; r.z = v1.z -
 v2.z; }
// vector clamp
inline void v_clamp(vec3 &v, double min, double max)
{
  v.x = clamp(v.x,min,max);
  v.y = clamp(v.y,min,max);
  v.z = clamp(v.z,min,max);
}

#endif
```

walk.cc

```c
#include "camera.h"
#include "vector3d.h"
#include "mandelbulb.h"
#include "mandelbox.h"
#include "color.h"
#include "3d.h"
```

```
#include "camera.h"
#include "renderer.h"
#include "walk.h"

#include <stdio.h>

// Get the next frame to render
// Computes an orbit around the bulb or box and reinitializes camera to point at cen
ter
// The computed orbit is a spiral following a circular path in the x-y plane with a
decreasing z
// See documentation for full algorithm description

#ifdef BULB
    extern double rayMarch(const int maxRaySteps, const float maxDistance,
      const float escape_time, const float power, const int num_iter,
      const vec3 &from, const vec3  &direction, double eps, pixelData& pix_data);

    extern double DE(const vec3 &p0,
        const float escape_time, const float power, const int num_iter);

#else //BOX
    extern double rayMarch(const int maxRaySteps, const float maxDistance,
      const int num_iter, const float rMin, const float rFixed, const float escape_t
ime, const float scale,
      const vec3 &from, const vec3  &direction, double eps, pixelData& pix_data);

    extern double DE(const vec3 &p0, const int num_iter, const float rMin,
        const float rFixed, const float escape_time, const float scale, double c1, d
ouble c2);
#endif

double VECTOR_OPTIONS [4] = {sqrt(1.0/(double)3.0), -sqrt(1.0/(double)3.0), (double)
1, (double)-1};
vec3 directions [28];

#ifdef BULB
void walk(CameraParams *camera_history,
            RenderParams *renderer_params,
            MandelBulbParams *bulb_params,
            int verbose, int frame)
#else
void walk(CameraParams *camera_history,
            RenderParams *renderer_params,
            MandelBoxParams *box_params,
```

```
            int verbose, int frame)
#endif
{
    // full rotation in x,y every 500*pi frames
    double inclination = frame/500.0;
    // normalize z range from 1 to -1 for 7200 frames
    double correction = frame / 3600.0;

    camera_history[frame + 1].camPos[0] = cos(inclination);
    camera_history[frame + 1].camPos[1] = sin(inclination);
    camera_history[frame + 1].camPos[2] = 1 - correction;

    // set camera for the next frame
    init3D(&camera_history[frame + 1], renderer_params);
}
```

walk.h

```c
#ifndef _walk_H
#define _walk_h
#include "camera.h"

#define PRINTVEC(vec, end) ( printf("(%f, %f, %f)%s", vec.x, vec.y, vec.z, end) )

#ifdef BULB
void walk(CameraParams *camera_history,
          RenderParams *renderer_params,
          MandelBulbParams *bulb_params,
          int verbose, int frame);
#else
void walk(CameraParams *camera_history,
          RenderParams *renderer_params,
          MandelBoxParams *box_params,
          int verbose, int frame);
#endif

#endif
```