TMR4160 Project Report

Tormod Gjeitnes Hellen

April 16, 2015

Introduction

The task given was to numerically solve the Poisson equation under Dirichlet and Neumann boundary conditions. The Dirichlet case was pretty straightforward to implement and performed pretty well on first try. In the Neumann case interperting the boundary condition (that the derivative normally over the boundary is 0) proved very challenging and frustrating.

Another source of error was that arrays are 1-indexed, which matters when you're trying to determine the coordinates from the matrix index.

Background

The problem presented is to numerically solve the two-dimensional Poisson equation $\nabla^2\varphi := \frac{\delta^2\varphi}{\delta x^2} + \frac{\delta^2\varphi}{\delta y^2} = f(x,y)$(find φ) under three different conditions

1. With $f(x,y) = 1$ generally and $\varphi(x,y) = \frac{1}{4}(x^2 + y^2)$ on the boundary(Dirichlet condition)

2. With $f(x,y) = 12 - 12x - 12y$ generally, $\varphi(0,0) = 0$ and $\frac{d\varphi}{dn} = 0$ on the boundary(Neumann condition), where n is a vector normal to the boundary.

3. With $f(x,y) = (6 - 12x)(3y^2 - 2y^3) + (3x^2 - 2x^3)(6 - 12y)$ generally, $\varphi(0,0) = 0$ and $\frac{d\varphi}{dn} = 0$ on the boundary(Neumann condition), where n is a vector normal to the boundary.

We were to use the discrete abstraction $\nabla^2\varphi := (\varphi(x+h,y) + \varphi(x-h,y) + \varphi(x,y+h) + \varphi(x,y-h) - 4\varphi(x,y))/h^2 = f(x,y)$. We were permitted to calculate over a quadratic field with points divided up uniformly in both directions.

Analysis and Design

Starting from the discrete equation we were allowed to use I decided to use relaxation, as it's relatively straightforward and performs well under most circumstances.

In order to use a relaxation method we need an update rule. Taking a hint as to the desired result from Wikipedia's article on relaxation we perform a transformation:

$$
\begin{aligned}
(\varphi(x+h,y) + \varphi(x-h,y) + \varphi(x,y+h) + \varphi(x,y-h) - 4\varphi(x,y))/h^2 &= f(x,y) \qquad (1)\\
\varphi(x+h,y) + \varphi(x-h,y) + \varphi(x,y+h) + \varphi(x,y-h) - 4\varphi(x,y) &= h^2 f(x,y)\\
\varphi(x+h,y) + \varphi(x-h,y) + \varphi(x,y+h) + \varphi(x,y-h) - h^2 f(x,y) &= 4\varphi(x,y)\\
(\varphi(x+h,y) + \varphi(x-h,y) + \varphi(x,y+h) + \varphi(x,y-h) - h^2 f(x,y))/4 &= \varphi(x,y)
\end{aligned}
$$

Using the relaxation strategy, we repeatedly set $\varphi(x,y)$ to be $(\varphi(x+h,y) + \varphi(x-h,y) + \varphi(x,y+h) + \varphi(x,y-h) - h^2 f(x,y))/4$ until some convergence criterium is fulfilled.

So far so good. And for the Dirichlet boundary condition, we don't need any more analytics than this. The solution is straightforward. However, now remains the thorny issue on how to best enforce $\frac{d\varphi}{dn} = 0$ for the Neumann boundary conditions.

Neumann Boundary Condition Enforcing

The first part $(\varphi(0,0) = 0)$ is straightforward, as we let that point be zero and never change it. There are some optimizations to be done near the origin, but they are just Enforcing $\frac{d\varphi}{dn} = 0$, however, proved very challenging. There are two particular cases that needs consideration:

1. The sides of the square

2. The corners of the square

Let ut consider each in isolation.

Corner options:

1. Designate the point nearest to the corner that can still be determined by the relaxation rule as special. Set the corner and the two neighboring points equal to this one. Let (x,y) be the top right corner

$$
\begin{aligned}
\varphi(x,y) &= \varphi(x-h,y-h)\\
\varphi(x-h,y) &= \varphi(x-h,y-h)\\
\varphi(x,y-h) &= \varphi(x-h,y-h)
\end{aligned}
$$

2. Since $\frac{d\varphi}{dn} = 0$, we can assume that φ does not change much between the corner and its neighbors. We simply average (for top right corner: $(x,y) = (max,max)$):

$$
\varphi(max,max) = (\varphi(max-1,max) + \varphi(max,max-1)\varphi(max-1,max-1))/3
$$

   - In the same way, we can say that the points near the origin will have φ be pretty close to zero and we can set this at the start of each iteration:

$$
\begin{aligned}
\varphi(0,1) &= 0\\
\varphi(1,0) &= 0\\
\varphi(1,1) &= 0
\end{aligned}
$$

   Let this be the RESET strategy

3. Approximate $\varphi(x,y)$ to be quadratic in the corner. Since $\frac{d\varphi}{dn} = 0$ we get that the two neighbours per direction of the boundary point are equal to each other, independent of the border point itself. Assuming (x,y) is in the top right corner $((x,y) = (max, max))$:

$$
\begin{aligned}
(\varphi(x+h,y) + \varphi(x-h,y) + \varphi(x,y+h) + \varphi(x,y-h) - 4\varphi(x,y))/h^2 &= f(x,y) \qquad (2)\\
(\varphi(x-h,y) + \varphi(x-h,y) + \varphi(x,y-h) + \varphi(x,y-h) - 4\varphi(x,y))/h^2 &= f(x,y)\\
(2\varphi(x-h,y) + 2\varphi(x,y-h) - 4\varphi(x,y))/h^2 &= f(x,y)\\
(2\varphi(x-h,y) + 2\varphi(x,y-h) - h^2 f(x,y))/4 &= \varphi(x,y)
\end{aligned}
$$

- We can use this to special effect on the origin, since we know that $\varphi(0,0) = 0$:

$$
\begin{aligned}
(2\varphi(h,0) + 2\varphi(0,h) - h^2 f(0,0))/4 &= 0\\
2\varphi(h,0) + 2\varphi(0,h) - h^2 f(0,0) &= 0\\
\varphi(h,0) &= (h^2 f(0,0) - 2\varphi(0,h))/2\\
\varphi(0,h) &= (h^2 f(0,0) - 2\varphi(h,0))/2
\end{aligned}
$$

This is circular, though, and makes the two right-angle neighbors to the origin completely independent of the rest of the matrix, which is clearly unfortunate. It can be used as a hint at the start of each iteration, where we can set (0,h) and (h,0) using each other's pre-set values. It can also be used for $\varphi(h,h)$:

$$
\varphi(h,h) = (\varphi(2h,h) + (h^2 f(0,0) - 2\varphi(h,0))/2 + \varphi(h,2h) + (h^2 f(0,0) - 2\varphi(0,h))/2 - h^2 f(x,y))/4
$$

For further discussion, this will be called the ORIGINDEMAND strategy.

Side options:

1. Approximate $\varphi(x,y)$ to be linear on the border. Since $\frac{d\varphi}{dn} = 0$ we get that the two neighbours in the n direction of a boundary point are equal to that point. This changes the math from (1) a bit. Assuming (x,y) is on the left border$(x = 0)$:

$$
\begin{aligned}
(\varphi(x+h,y) + \varphi(x-h,y) + \varphi(x,y+h) + \varphi(x,y-h) - 4\varphi(x,y))/h^2 &= f(x,y) \qquad (3)\\
(\varphi(x,y) + \varphi(x,y) + \varphi(x,y+h) + \varphi(x,y-h) - 4\varphi(x,y))/h^2 &= f(x,y)\\
(\varphi(x,y+h) + \varphi(x,y-h) - 2\varphi(x,y))/h^2 &= f(x,y)\\
\varphi(x,y+h) + \varphi(x,y-h) - 2\varphi(x,y) &= h^2 f(x,y)\\
\varphi(x,y+h) + \varphi(x,y-h) - h^2 f(x,y) &= 2\varphi(x,y)\\
(\varphi(x,y+h) + \varphi(x,y-h) - h^2 f(x,y))/2 &= \varphi(x,y)
\end{aligned}
$$

- Optionally, we can say that, in order for this to be consistent with how the in-memory matrix actually looks, the boundary point's neighbor inside the matrix is updated along with the boundary point. This means that, for $x = 0$, after updating $\varphi(x+h,y) == \varphi(x,y)$.

2. Approximate $\varphi(x,y)$ to be quadratic on the border. Since $\frac{d\varphi}{dn} = 0$ we get that the two neighbours in the n direction of a boundary point are equal to each other, independent of the border point itself. Assuming (x,y) is on the left border$(x = 0)$:

$$
\begin{aligned}
(\varphi(x+h,y) + \varphi(x-h,y) + \varphi(x,y+h) + \varphi(x,y-h) - 4\varphi(x,y))/h^2 &= f(x,y) \qquad (4)\\
(\varphi(x+h,y) + \varphi(x+h,y) + \varphi(x,y+h) + \varphi(x,y-h) - 4\varphi(x,y))/h^2 &= f(x,y)\\
(2\varphi(x+h,y) + \varphi(x,y+h) + \varphi(x,y-h) - 4\varphi(x,y))/h^2 &= f(x,y)\\
2\varphi(x-h,y) + \varphi(x,y+h) + \varphi(x,y-h) - 4\varphi(x,y) &= h^2 f(x,y)\\
2\varphi(x+h,y) + \varphi(x,y+h) + \varphi(x,y-h) - h^2 f(x,y) &= 4\varphi(x,y)\\
(2\varphi(x+h,y) + \varphi(x,y+h) + \varphi(x,y-h) - h^2 f(x,y))/4 &= \varphi(x,y)
\end{aligned}
$$

Analytical Solutions

For the first task (the Dirichlet one) we need $\nabla^2\varphi = 1$in general and $\varphi(x,y) = \frac{1}{4}(x^2 + y^2)$ on the boundary. Turns out that $\varphi(x,y) = \frac{1}{4}(x^2 + y^2)$ satisfies both.

For the second task (first Neumann one) we need $\nabla^2\varphi = 12 - 12x - 12y$ generally, $\varphi(0,0) = 0$ and $\frac{d\varphi}{dn} = 0$ on boundary. With some trial and error, it can be determined that $3x^2 + 3y^2 - 2x^3 - 2y^3$ satisfies those.

For the third task (second Neumann one) we need $\nabla^2\varphi = (6 - 12x)(3y^2 - 2y^3) + (3x^2 - 2x^3)(6 - 12y)$ generally, $\varphi(0,0) = 0$ and $\frac{d\varphi}{dn} = 0$ on boundary. This seems like a very hard nut to crack until you realise that:

$$(6 - 12x)(3y^2 - 2y^3) + (3x^2 - 2x^3)(6 - 12y) = \frac{\delta^2 i}{\delta x^2}j + i\frac{\delta^2 j}{\delta y^2} \tag{5}$$

and that

$$\nabla^2(ij) \quad = \quad \frac{\delta^2 ij}{\delta x^2} + \frac{\delta^2 ij}{\delta y^2} \tag{6}$$

$$\nabla^2(ij) \quad = \quad \frac{\delta^2 i}{\delta x^2}j + i\frac{\delta^2 j}{\delta x^2} + \frac{\delta^2 i}{\delta y^2}j + i\frac{\delta^2 j}{\delta y^2} + 2\frac{\delta i}{\delta x}\frac{\delta j}{\delta x} + 2\frac{\delta i}{\delta y}\frac{\delta j}{\delta y} \tag{7}$$

Here all the terms reduce to 0 except $\frac{\delta^2 i}{\delta x^2}j + i\frac{\delta^2 j}{\delta y^2}$, which means, together with the other requirements, that $\varphi(x, y) = (3y^2 - 2y^3)(3x^2 - 2x^3)$.

Memory use

No memory is allocated dynamically and only the matrix holding the values itself has a size that depends on inter-point distance. In terms of big-O notation, memory use is $O(1/h^2)$, where h is the inter-point distance. An interesting note is that, as the matrix is the main source of memory consumption, setting real to be 8 bytes means memory consumption roughly doubles.

Convergence criterium

There were many possible ones to consider

1. Setting a minimum permitted ratio of current average change to previous average change

2. Setting a minimum permitted ratio of current maximum change to previous maximum change

3. For Neumann: Using the fact that we know points close to origin to be close to zero, we can say that we want (1,1) to be closer to zero than a certain difference.

4. Observing that for 1 and 2, the changes cannot be permitted to rise, they can be combined and measure both of these and let none of them rise

5. Modifying 4, the all time lows of average change and maximum change is tracked, and a minimum permitted ratio for each current of these two to the all time low developed. If both are breached at the same time, the algorithm is said to have converged.

Implementation

List of functions and description of each:

F	the right side of the poisson equation
ANALYTICAL	computes the analytical solution
SIMPLEESTIMATE	computes a new estimate for the value of (x*h,y*h). Not valid on the boundary
NEWESTIMATE	computes the value of ((x-1)*h,(y-h)*h). Valid everywhere.
HIGHESTCHANGEFUN	returns the gighest change in (phi-)value given a new old value, new assignment and the previous highest
LOWCHANGE	returns the lesser of two variables. Intended to compare change magnitudes to determine the lowest o
ANALYTICALERROR	computes the analytically determined average squared error

Here is the most interesting code, which is for the Neumann case

```
FUNCTION NEWESTIMATE(X,Y,H,PHIS,LENGTH)
  !computes the value of ((x-1)*h,(y-h)*h). Valid everywhere.
  INTEGER X
  INTEGER Y
  REAL H
  INTEGER LENGTH
  REAL NEWVALUE
  REAL :: PHIS(LENGTH,LENGTH)
  REAL NEWESTIMATE
  REAL SIMPLEESTIMATE
  REAL F
  REAL :: ORIGINDEMAND
  IF (X==1 .AND. Y==1) THEN
    TWO_ONE=(F(1,1,H)**2-2*PHIS(1,2))/2.0
    ONE_TWO=(F(1,1,H)**2-2*PHIS(2,1))/2.0
    PHIS(1,2)=ONE_TWO
    PHIS(2,1)=TWO_ONE
    NEWESTIMATE=0.0
  ELSE IF (X==2 .AND. Y==2) THEN
    TWO_ONE=(F(1,1,H)**2-2*PHIS(1,2))/2.0
    ONE_TWO=(F(1,1,H)**2-2*PHIS(2,1))/2.0
    NEWESTIMATE = (1.0/4.0)*(PHIS(X+1,Y)+PHIS(X,Y+1)+ONE_TWO+TWO_ONE-H*H*F(X,Y,H))
  ELSE IF (X==1 .AND. Y==LENGTH) THEN
    NEWESTIMATE = (PHIS(2,LENGTH-1) + PHIS(2,LENGTH) + PHIS(1,LENGTH-1))/3.0
  ELSE IF (X==LENGTH .AND. Y==1) THEN
    NEWESTIMATE = (PHIS(LENGTH-1,2) + PHIS(LENGTH,2) + PHIS(LENGTH-1,1))/3.0
  ELSE IF (X==LENGTH .AND. Y==LENGTH) THEN
    NEWESTIMATE = (PHIS(LENGTH-1,LENGTH-1) + PHIS(LENGTH-1,LENGTH) + PHIS(LENGTH,LENGTH-1))/3.0
  ELSE IF (X==1) THEN
    NEWVALUE = (1.0/4.0)*(2*PHIS(X+1,Y)+PHIS(X,Y+1)+PHIS(X,Y-1)-H*H*F(X,Y,H))
    NEWESTIMATE = NEWVALUE
  ELSE IF (X==LENGTH) THEN
    NEWVALUE = (1.0/4.0)*(PHIS(X,Y+1)+2*PHIS(X-1,Y)+PHIS(X,Y-1)-H*H*F(X,Y,H))
    NEWESTIMATE = NEWVALUE
  ELSE IF (Y==1) THEN
    NEWVALUE = (1.0/4.0)*(PHIS(X+1,Y)+2*PHIS(X,Y+1)+PHIS(X-1,Y)-H*H*F(X,Y,H))
    NEWESTIMATE = NEWVALUE
  ELSE IF (Y==LENGTH) THEN
```

```
      NEWVALUE = (1.0/4.0)*(PHIS(X+1,Y)+PHIS(X-1,Y)+2*PHIS(X,Y-1)-H*H*F(X,Y,H))
      NEWESTIMATE = NEWVALUE
   ELSE
      NEWESTIMATE = SIMPLEESTIMATE(X,Y,H,PHIS,LENGTH)
   END IF
   RETURN
END FUNCTION


FUNCTION SIMPLEESTIMATE(X,Y,H,PHIS,LENGTH)
   !computes a new estimate for the value of (x*h,y*h). Not valid on the boundary
   INTEGER X,Y
   REAL H
   REAL SIMPLEESTIMATE
   INTEGER LENGTH
   REAL F
   REAL :: PHIS(LENGTH,LENGTH)
   SIMPLEESTIMATE = (1.0/4.0)*(PHIS(X+1,Y)+PHIS(X,Y+1)+PHIS(X-1,Y)+PHIS(X,Y-1)-H*H*F(X,Y,H))
   RETURN
END FUNCTION
```
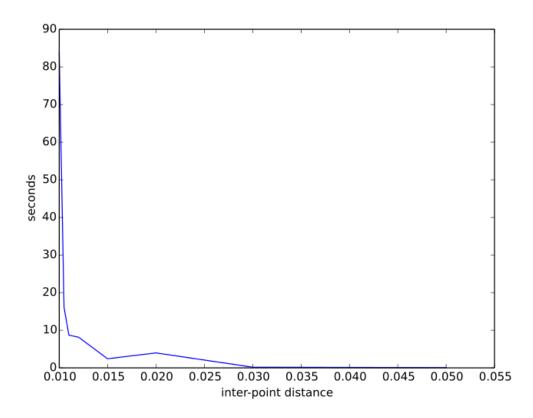
And the main loop:

```
NUMITERATIONS = 0
!the actual computation is performed here
DO WHILE  ((LOWHIGHESTCHANGE/HIGHESTCHANGE) > 1.00001 .OR. (LOWAVGCHANGE/AVGCHANGE) > 1.00001)
   NUMITERATIONS = NUMITERATIONS + 1
   LOWAVGCHANGE = LOWCHANGE(LOWAVGCHANGE, AVGCHANGE)
   LOWHIGHESTCHANGE = LOWCHANGE(LOWHIGHESTCHANGE, HIGHESTCHANGE)
   HIGHESTCHANGE = 0.0
   AVGCHANGE = 0.0
   DO J=1,LENGTH
      DO I=1,LENGTH
         NEWVALUE = NEWESTIMATE(I,J,H,PHIS,LENGTH)
         HIGHESTCHANGE = HIGHESTCHANGEFUN(PHIS(I,J),NEWVALUE,HIGHESTCHANGE)
         AVGCHANGE = AVGCHANGE + ((PHIS(I,J)-NEWVALUE)**2)/SIZE
         PHIS(I,J) = NEWVALUE
      END DO
   END DO
END DO
```

The display code is almost entirely taken from the Mesa project's "array" example.

Testing

In the end it turned out that whether you approximates the phi at the boundary as being linear or quadratic has little effect. On, the corners, however, simple averaging gives markedly superior results. Setting the four points at the origin and closest to it to zero at each iteration start also improves performance. The ORIGINDEMAND strategy can be used with equally good results.

For small inter-point distances, the neumann programs are really slow. A possible reason is that the constant point at the origin must have the consequences of its value propagated through the matrix and this requires more iterations, even as each iteration becomes more costly.

Conclusion, Evaluation and Further Work

The algorithm is way too sensitive to the initial state of the matrix representing the φ values. It's a troublesome problem, but it's hard to know what to do about it. Since the pseudorandom numbers used to populate the matrix are deterministic and equal at each invocation, some inter-point distances(h) causes the algorithm to choke, producing incorrect results, while others prompt a speedy and correct response.

A possible way to speed up the algorithm would be to first do a run with large inter-point distance, and then use the result to initialize the bigger matrix. Unfortunately this has not been tried during the work with this report.

Appendix A

User manual

To compile and run the Neumann case, use "gfortran neumannFinal.f95 -fimplicit-none -O3 -o neumannFinal -fdefault-real-8 && time ./neumannFinal"

Remember to comment in or out the correct f and analytical functions!

To compile and run the Dirichlet case, use "gfortran dirichlet.f95 -fimplicit-none -O3 -o dirichlet -fdefault-real-8 && time ./dirichlet"

To compile the display program use "clang -o array array.c shaderutil.c -lGL -lglut -lm -lGLEW -O3". To run it, be in the same folder as the result.txt file and use "./array" or "./array/array" depending on where the executable is.

Appendix B

Code listing

Neumann(neumannFinal.f95):

```fortran
! gfortran neumannFinal.f95 -fimplicit-none -O3 -o neumannFinal -fdefault-real-8 && time ./neumannFina

      FUNCTION F(X,Y,H)
        !the right side of the poisson equation
        INTEGER X,Y
        REAL H
        REAL F
        REAL :: XC
        REAL :: YC
        XC = (X-1)*H
        YC = (Y-1)*H
        !F=12-12*XC-12*YC
        F=(6-12*XC)*(3*YC**2-2*YC**3) + (3*XC**2-2*XC**3)*(6-12*YC)
        RETURN
      END FUNCTION

      FUNCTION ANALYTICAL(X,Y,H)
        !computes the analytical solution
        REAL ANALYTICAL
        INTEGER :: X
        INTEGER :: Y
        REAL :: H
        REAL :: XC
        REAL :: YC
        XC = (X-1)*H
        YC = (Y-1)*H
        !ANALYTICAL = 3*XC**2 + 3*YC**2 - 2*XC**3 - 2*YC**3
        ANALYTICAL = (3*YC**2 - 2*YC**3)*(3*XC**2 - 2*XC**3)
        RETURN
      END FUNCTION

      FUNCTION NEWESTIMATE(X,Y,H,PHIS,LENGTH)
        !computes the value of ((x-1)*h,(y-h)*h). Valid everywhere.
        INTEGER X
        INTEGER  Y
        REAL H
        INTEGER LENGTH
        REAL NEWVALUE
        REAL :: PHIS(LENGTH,LENGTH)
        REAL NEWESTIMATE
        REAL SIMPLEESTIMATE
        REAL F
        REAL :: ORIGINDEMAND
```

11

```
      REAL :: TWO_ONE
      REAL :: ONE_TWO
     IF (X==1 .AND. Y==1) THEN
        TWO_ONE=(F(1,1,H)**2-2*PHIS(1,2))/2.0
        ONE_TWO=(F(1,1,H)**2-2*PHIS(2,1))/2.0
        PHIS(1,2)=ONE_TWO
        PHIS(2,1)=TWO_ONE
        NEWESTIMATE=0.0
     ELSE IF (X==2 .AND. Y==2) THEN
        TWO_ONE=(F(1,1,H)**2-2*PHIS(1,2))/2.0
        ONE_TWO=(F(1,1,H)**2-2*PHIS(2,1))/2.0
        NEWESTIMATE = (1.0/4.0)*(PHIS(X+1,Y)+PHIS(X,Y+1)+ONE_TWO+TWO_ONE-H*H*F(X,Y,H))
     ELSE IF (X==1 .AND. Y==LENGTH) THEN
        NEWESTIMATE = (PHIS(2,LENGTH-1) + PHIS(2,LENGTH) + PHIS(1,LENGTH-1))/3.0
     ELSE IF (X==LENGTH .AND. Y==1) THEN
        NEWESTIMATE = (PHIS(LENGTH-1,2) + PHIS(LENGTH,2) + PHIS(LENGTH-1,1))/3.0
     ELSE IF (X==LENGTH .AND. Y==LENGTH) THEN
        NEWESTIMATE = (PHIS(LENGTH-1,LENGTH-1) + PHIS(LENGTH-1,LENGTH) + PHIS(LENGTH,LENGTH-1))/3.0
     ELSE IF (X==1) THEN
        NEWVALUE = (1.0/4.0)*(2*PHIS(X+1,Y)+PHIS(X,Y+1)+PHIS(X,Y-1)-H*H*F(X,Y,H))
        NEWESTIMATE = NEWVALUE
     ELSE IF (X==LENGTH) THEN
        NEWVALUE = (1.0/4.0)*(PHIS(X,Y+1)+2*PHIS(X-1,Y)+PHIS(X,Y-1)-H*H*F(X,Y,H))
        NEWESTIMATE = NEWVALUE
     ELSE IF (Y==1) THEN
        NEWVALUE = (1.0/4.0)*(PHIS(X+1,Y)+2*PHIS(X,Y+1)+PHIS(X-1,Y)-H*H*F(X,Y,H))
        NEWESTIMATE = NEWVALUE
     ELSE IF (Y==LENGTH) THEN
        NEWVALUE = (1.0/4.0)*(PHIS(X+1,Y)+PHIS(X-1,Y)+2*PHIS(X,Y-1)-H*H*F(X,Y,H))
        NEWESTIMATE = NEWVALUE
     ELSE
        NEWESTIMATE = SIMPLEESTIMATE(X,Y,H,PHIS,LENGTH)
     END IF
     RETURN
END FUNCTION


FUNCTION SIMPLEESTIMATE(X,Y,H,PHIS,LENGTH)
    !computes a new estimate for the value of (x*h,y*h). Not valid on the boundary
    INTEGER X,Y
    REAL H
    REAL SIMPLEESTIMATE
    INTEGER LENGTH
    REAL F
    REAL :: PHIS(LENGTH,LENGTH)
    SIMPLEESTIMATE = (1.0/4.0)*(PHIS(X+1,Y)+PHIS(X,Y+1)+PHIS(X-1,Y)+PHIS(X,Y-1)-H*H*F(X,Y,H))
    RETURN
END FUNCTION

FUNCTION HIGHESTCHANGEFUN(OLD,NEW,PREVHIGHEST)
    !returns the gighest change in (phi-)value given a new old value, new assignment and the prev
    REAL :: OLD
    REAL :: NEW
    REAL :: PREVHIGHEST
    REAL :: CHANGE
    REAL HIGHESTCHANGEFUN
    CHANGE = ABS(OLD-NEW)
    IF (CHANGE .GE. PREVHIGHEST) THEN
      HIGHESTCHANGEFUN = CHANGE
```

```
      ELSE
         HIGHESTCHANGEFUN = PREVHIGHEST
      END IF
      RETURN
END FUNCTION

FUNCTION LOWCHANGE(OLDLOWEST, LASTLOW)
   !returns the lesser of two variables. Intended to compare change magnitudes to determine the l
   REAL LOWCHANGE
   REAL OLDLOWEST
   REAL LASTLOW
   IF (OLDLOWEST .LT. LASTLOW) THEN
      LOWCHANGE = OLDLOWEST
   ELSE
      LOWCHANGE = LASTLOW
   END IF
END FUNCTION

FUNCTION ANALYTICALERROR(PHIS,LENGTH,H,SIZE)
   !computes the analytically determined average squared error
   INTEGER SIZE
   INTEGER LENGTH
   REAL H
   REAL :: PHIS(LENGTH,LENGTH)
   REAL ANALYTICAL
   REAL AVGERROR
   INTEGER I,J
   REAL ANALYTICALERROR
   AVGERROR = 0.0
   DO J=1,LENGTH
      DO I=1,LENGTH
      AVGERROR = AVGERROR + ((PHIS(I,J)−ANALYTICAL(I,J,H))**2)/SIZE
      END DO
   END DO
   ANALYTICALERROR = AVGERROR
   RETURN
END FUNCTION

PROGRAM SOLVER
   REAL, PARAMETER :: H = 0.0105
   INTEGER, PARAMETER :: LENGTH = (1.0/H)+1
   INTEGER, PARAMETER :: SIZE = LENGTH*LENGTH
   REAL :: PHIS(LENGTH, LENGTH)
   INTEGER, PARAMETER :: out_unit=20
   REAL :: HIGHESTCHANGE
   REAL :: NEWVALUE
   REAL SIMPLEESTIMATE
   REAL NEWESTIMATE
   REAL ANALYTICAL
   INTEGER I,J
   REAL HIGHESTCHANGEFUN
   REAL LOWHIGHESTCHANGE
   REAL AVGCHANGE
   REAL LOWAVGCHANGE
   REAL LOWCHANGE
   REAL AVGERROR
   REAL NUMAVGERROR
   INTEGER NUMITERATIONS
   REAL ANALYTICALERROR
```

```fortran
REAL :: r(5,5)
INTEGER :: SEED
LOWAVGCHANGE = 20000.0
AVGCHANGE = 10000.0
LOWHIGHESTCHANGE = 20000.0
HIGHESTCHANGE = 10000.0

WRITE (*,*) "LENGTH IS ", LENGTH
!a friendly reminder that the real coordinate is (x-1)*h, not x*h
WRITE (*,*) "(LENGTH-1)*H IS ", ((LENGTH-1)*H)

DO I=1,LENGTH
  DO J=1,LENGTH
     PHIS(I,J) = RAND(SEED)*10.0
  END DO
END DO

NUMITERATIONS = 0
!the actual computation is performed here
DO WHILE      ((LOWHIGHESTCHANGE/HIGHESTCHANGE) > 1.1 .OR. (LOWAVGCHANGE/AVGCHANGE) > 1.000008
  NUMITERATIONS = NUMITERATIONS + 1
  LOWAVGCHANGE = LOWCHANGE(LOWAVGCHANGE, AVGCHANGE)
  LOWHIGHESTCHANGE = LOWCHANGE(LOWHIGHESTCHANGE, HIGHESTCHANGE)
  HIGHESTCHANGE = 0.0
  AVGCHANGE = 0.0
  DO J=1,LENGTH
    DO I=1,LENGTH
        NEWVALUE = NEWESTIMATE(I,J,H,PHIS,LENGTH)
        HIGHESTCHANGE = HIGHESTCHANGEFUN(PHIS(I,J),NEWVALUE,HIGHESTCHANGE)
        AVGCHANGE = AVGCHANGE + ((PHIS(I,J)-NEWVALUE)**2)/SIZE
        PHIS(I,J) = NEWVALUE
    END DO
  END DO
  !WRITE (*,*) "HIGHESTCHANGE IS ", HIGHESTCHANGE
  !WRITE (*,*) "AVGCHANGE IS ", AVGCHANGE
  !WRITE (*,*) "LOWAVGCHANGE IS ", LOWAVGCHANGE
  !WRITE (*,*) "LOWHIGHESTCHANGE IS ", LOWHIGHESTCHANGE
  !WRITE (*,*) ""
END DO

WRITE (*,*) "ratio", (LOWHIGHESTCHANGE/HIGHESTCHANGE) > 1.00000000
WRITE (*,*) "ratio", LOWHIGHESTCHANGE/HIGHESTCHANGE
WRITE (*,*) "ratio", (LOWAVGCHANGE/AVGCHANGE) > 1.00000000
WRITE (*,*) "ratio", LOWAVGCHANGE/AVGCHANGE



WRITE (*,*) "NUMITERATIONS IS ", NUMITERATIONS
!finds and prints the numerically determined error
NUMAVGERROR = 0.0
DO I=1,LENGTH
  DO J=1,LENGTH
  NUMAVGERROR = NUMAVGERROR + ((PHIS(I,J)-NEWESTIMATE(I,J,H,PHIS,LENGTH))**2)/SIZE
  END DO
END DO
WRITE (*,*) "AVERAGE SQUARE NUMERICALLY ESTIMATED ERROR IS ", NUMAVGERROR

!performance metrics. Analytically determined error is printed
AVGERROR = 0.0
```

14

```
       DO I=1,LENGTH
         DO J=1,LENGTH
         AVGERROR = AVGERROR + ((PHIS(I,J)-ANALYTICAL(I,J,H))**2)/SIZE
         END DO
       END DO
       WRITE (*,*) "AVERAGE SQUARE ANALYTICALLY DETERMINED ERROR IS ", AVGERROR

       !writes results to file
       open (unit=out_unit,file="results.txt",action="write",status="replace")
       WRITE (OUT_UNIT,'(I4)') LENGTH
       WRITE (OUT_UNIT,'(I4)') LENGTH
       DO I=1,LENGTH
         DO J=1,LENGTH
             WRITE (OUT_UNIT,'(F0.5)') PHIS(I,J)
         END DO
       END DO
       close (out_unit)
     END
```

Dirichlet (dirichlet.f95):

```
!gfortran dirichlet.f95 -fimplicit-none -O3 -o dirichlet -fdefault-real-8 && time ./dirichlet

     FUNCTION F(X,Y,H)
       !the right side of the poisson equation
       INTEGER X,Y
       REAL H
       REAL F
       F=1
       RETURN
     END FUNCTION

     FUNCTION ANALYTICAL(X,Y,H)
       !computes the analytical solution
       REAL ANALYTICAL
       INTEGER :: X
       INTEGER :: Y
       REAL :: H
       REAL :: XC
       REAL :: YC
       XC = (X-1)*H
       YC = (Y-1)*H
       ANALYTICAL = 0.25*(XC**2+YC**2)
       RETURN
     END FUNCTION

     FUNCTION NEWESTIMATE(X,Y,H,PHIS,LENGTH)
       !computes the value of ((x-1)*h,(y-h)*h). Valid everywhere.
       INTEGER X
       INTEGER Y
       REAL H
       INTEGER LENGTH
       REAL NEWVALUE
       REAL :: PHIS(LENGTH,LENGTH)
       REAL NEWESTIMATE
       REAL SIMPLEESTIMATE
       REAL F
       REAL :: ORIGINDEMAND
       REAL :: XC
       REAL :: YC
```

15

```fortran
      XC = (X-1)*H
      YC = (Y-1)*H
      IF (X==1 .OR. Y==1 .OR. X==LENGTH .OR. Y==LENGTH) THEN
        NEWESTIMATE = 0.25*(XC**2+YC**2)
      ELSE
        NEWESTIMATE = SIMPLEESTIMATE(X,Y,H,PHIS,LENGTH)
      END IF
      RETURN
END FUNCTION


FUNCTION SIMPLEESTIMATE(X,Y,H,PHIS,LENGTH)
   !computes a new estimate for the value of (x*h,y*h). Not valid on the boundary
   INTEGER X,Y
   REAL H
   REAL SIMPLEESTIMATE
   INTEGER LENGTH
   REAL F
   REAL :: PHIS(LENGTH,LENGTH)
   SIMPLEESTIMATE = (1.0/4.0)*(PHIS(X+1,Y)+PHIS(X,Y+1)+PHIS(X-1,Y)+PHIS(X,Y-1)-H*H*F(X,Y,H))
   RETURN
END FUNCTION

FUNCTION HIGHESTCHANGEFUN(OLD,NEW,PREVHIGHEST)
   !returns the gighest change in (phi-)value given a new old value, new assignment and the prev
   REAL :: OLD
   REAL :: NEW
   REAL :: PREVHIGHEST
   REAL :: CHANGE
   REAL HIGHESTCHANGEFUN
   CHANGE = ABS(OLD-NEW)
   IF (CHANGE .GE. PREVHIGHEST) THEN
      HIGHESTCHANGEFUN = CHANGE
   ELSE
      HIGHESTCHANGEFUN = PREVHIGHEST
   END IF
   RETURN
END FUNCTION

FUNCTION LOWCHANGE(OLDLOWEST, LASTLOW)
   !returns the lesser of two variables. Intended to compare change magnitudes to determine the l
   REAL LOWCHANGE
   REAL OLDLOWEST
   REAL LASTLOW
   IF (OLDLOWEST .LT. LASTLOW) THEN
      LOWCHANGE = OLDLOWEST
   ELSE
      LOWCHANGE = LASTLOW
   END IF
END FUNCTION

FUNCTION ANALYTICALERROR(PHIS,LENGTH,H,SIZE)
   !computes the analytically determined average squared error
   INTEGER SIZE
   INTEGER LENGTH
   REAL H
   REAL :: PHIS(LENGTH,LENGTH)
   REAL ANALYTICAL
   REAL AVGERROR
```

```
      INTEGER I , J
      REAL ANALYTICALERROR
      AVGERROR = 0.0
      DO J=1,LENGTH
        DO I=1,LENGTH
        AVGERROR = AVGERROR + ((PHIS(I,J)-ANALYTICAL(I,J,H))**2)/SIZE
        END DO
      END DO
      ANALYTICALERROR = AVGERROR
      RETURN
END FUNCTION

PROGRAM SOLVER
      REAL, PARAMETER :: H = 0.01
      INTEGER, PARAMETER :: LENGTH = (1.0/H)+1
      INTEGER, PARAMETER :: SIZE = LENGTH*LENGTH
      REAL :: PHIS(LENGTH, LENGTH)
      INTEGER, PARAMETER :: out_unit=20
      REAL :: HIGHESTCHANGE
      REAL :: NEWVALUE
      REAL SIMPLEESTIMATE
      REAL NEWESTIMATE
      REAL ANALYTICAL
      INTEGER I , J
      REAL HIGHESTCHANGEFUN
      REAL LOWHIGHESTCHANGE
      REAL AVGCHANGE
      REAL LOWAVGCHANGE
      REAL LOWCHANGE
      REAL AVGERROR
      REAL NUMAVGERROR
      INTEGER NUMITERATIONS
      REAL ANALYTICALERROR
      LOWAVGCHANGE = 20.0
      AVGCHANGE = 1.0
      LOWHIGHESTCHANGE = 20.0
      HIGHESTCHANGE = 10.0

      !a friendly reminder that what the real coordinate is (x-1)*h, not x*h
      WRITE (*,*) "(LENGTH-1)*H IS ", ((LENGTH-1)*H)

      DO I=1,LENGTH
        DO J=1,LENGTH
          PHIS(I,J) = RAND(0)*10
        END DO
      END DO

      NUMITERATIONS = 0
      !the actual computation is performed here
      DO WHILE  ((LOWHIGHESTCHANGE/HIGHESTCHANGE) > 1.00001 .OR. (LOWAVGCHANGE/AVGCHANGE) > 1.00001)
        NUMITERATIONS = NUMITERATIONS + 1
        LOWAVGCHANGE = LOWCHANGE(LOWAVGCHANGE, AVGCHANGE)
        LOWHIGHESTCHANGE = LOWCHANGE(LOWHIGHESTCHANGE, HIGHESTCHANGE)
        HIGHESTCHANGE = 0.0
        AVGCHANGE = 0.0
        DO J=1,LENGTH
          DO I=1,LENGTH
              NEWVALUE = NEWESTIMATE(I,J,H,PHIS,LENGTH)
              HIGHESTCHANGE = HIGHESTCHANGEFUN(PHIS(I,J),NEWVALUE,HIGHESTCHANGE)
```

17

```fortran
                AVGCHANGE = AVGCHANGE + ((PHIS(I,J)-NEWVALUE)**2)/SIZE
                PHIS(I,J) = NEWVALUE
            END DO
          END DO
        END DO



        WRITE (*,*) "NUMITERATIONS IS ", NUMITERATIONS
        !finds and prints the numerically determined error
        NUMAVGERROR = 0.0
        DO I=1,LENGTH
           DO J=1,LENGTH
           NUMAVGERROR = NUMAVGERROR + ((PHIS(I,J)-NEWESTIMATE(I,J,H,PHIS,LENGTH))**2)/SIZE
           END DO
        END DO
        WRITE (*,*) "AVERAGE SQUARE NUMERICALLY ESTIMATED ERROR IS ", NUMAVGERROR

        !performance metrics. Analytically determined error is printed
        AVGERROR = 0.0
        DO I=1,LENGTH
           DO J=1,LENGTH
           AVGERROR = AVGERROR + ((PHIS(I,J)-ANALYTICAL(I,J,H))**2)/SIZE
           END DO
        END DO
        WRITE (*,*) "AVERAGE SQUARE ANALYTICALLY DETERMINED ERROR IS ", AVGERROR

        !writes results to file
        open (unit=out_unit,file="results.txt",action="write",status="replace")
        WRITE (OUT_UNIT,'(I4)') LENGTH
        WRITE (OUT_UNIT,'(I4)') LENGTH
        DO I=1,LENGTH
           DO J=1,LENGTH
                WRITE (OUT_UNIT,'(F0.5)') PHIS(I,J)
           END DO
        END DO
        close (out_unit)
      END
```

Display(array.c):

```c
//clang -o array array.c shaderutil.c -lGL -lglut -lm -lGLEW -O3 && ./array

/**
 * Test variable array indexing in a vertex shader.
 * Brian Paul
 * 17 April 2009
 */

#include <assert.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <GL/glew.h>
#include "glut_wrap.h"
#include "shaderutil.h"

int LENGTH;
float *inm;
```

```
#define GLmaxdiff 2.0
float highest = 0.0;
float lowest = 0.0;

/**
 * The vertex position.z is used as a (variable) index into an
 * array which returns a new Z value.
 */
static const char *VertShaderText =
    "uniform sampler2D tex1; \n"
    "uniform float HeightArray[20]; \n"
    "void main() \n"
    "{ \n"
    "    vec4 pos = gl_Vertex; \n"
    "    int i = int(pos.z * 9.5); \n"
    "    pos.z = HeightArray[i]; \n"
    "    gl_Position = gl_ModelViewProjectionMatrix * pos; \n"
    "    gl_FrontColor = pos; \n"
    "} \n";

static const char *FragShaderText =
    "void main() \n"
    "{ \n"
    "    gl_FragColor = gl_Color; \n"
    "} \n";


static GLuint fragShader;
static GLuint vertShader;
static GLuint program;

static GLint win = 0;
static GLboolean Anim = GL_TRUE;
static GLboolean WireFrame = GL_TRUE;
static GLfloat xRot = -70.0f, yRot = 0.0f, zRot = 0.0f;


static void
Idle(void)
{
    zRot = 90 + glutGet(GLUT_ELAPSED_TIME) * 0.05;
    glutPostRedisplay();
}


/** z=f(x,y) */
static float
fz(float x, float y)
{
    return fabs(cos(1.5*x) + cos(1.5*y));
}

static float
getinm(float x, float y, float minx, float maxx, float miny, float maxy)
{
    int i = ((x+(-minx))/(maxx-minx))*LENGTH;
    int j = ((y+(-miny))/(maxy-miny))*LENGTH;
    int index = i*LENGTH+j;
    if(index<0){index=0;}
```

```c
    if (index>LENGTH*LENGTH){index=LENGTH*LENGTH−1;}
    //printf("x is %f, y is %f, i is %d, j is %d, index is %d\n",x,y,i,j,index);          //disabled
    return inm[index];
}


static void
DrawMesh(void)
{
    GLfloat xmin = −2.0, xmax = 2.0;
    GLfloat ymin = −2.0, ymax = 2.0;
    GLuint xdivs = 60, ydivs = 60;
    GLfloat dx = (xmax − xmin) / xdivs;
    GLfloat dy = (ymax − ymin) / ydivs;
    GLfloat ds = 1.0 / xdivs, dt = 1.0 / ydivs;
    GLfloat x, y, s, t;
    GLuint i, j;

    float scale = GLmaxdiff/(highest−lowest);

    y = ymin;
    t = 0.0;
    for (i = 0; i < ydivs; i++) {
        x = xmin;
        s = 0.0;
        glBegin(GL_QUAD_STRIP);
        for (j = 0; j < xdivs; j++) {
            float z0 = scale*getinm(x, y, xmin,xmax,ymin,ymax), z1 = scale*getinm(x, y + dy, xmin,xmax,y

            glTexCoord2f(s, t);
            glVertex3f(x, y, z0);

            glTexCoord2f(s, t + dt);
            glVertex3f(x, y + dy, z1);
            x += dx;
            s += ds;
        }
        glEnd();
        y += dy;
        t += dt;
    }

    /*y = ymin;
    t = 0.0;
    for (i = 0; i < ydivs; i++) {
        x = xmin;
        s = 0.0;
        glBegin(GL_QUAD_STRIP);
        for (j = 0; j < xdivs; j++) {
            float z0 = fz(x, y), z1 = fz(x, y + dy);

            glTexCoord2f(s, t);
            glVertex3f(x, y, z0);

            glTexCoord2f(s, t + dt);
            glVertex3f(x, y + dy, z1);
            x += dx;
            s += ds;
        }
```

```
        glEnd();
        y += dy;
        t += dt;
    }*/
}


static void
Redisplay(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    if (WireFrame)
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    else
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

    glPushMatrix();
    glRotatef(xRot, 1.0f, 0.0f, 0.0f);
    glRotatef(yRot, 0.0f, 1.0f, 0.0f);
    glRotatef(zRot, 0.0f, 0.0f, 1.0f);

    glPushMatrix();
    DrawMesh();
    glPopMatrix();

    glPopMatrix();

    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

    glutSwapBuffers();
}


static void
Reshape(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-1.0, 1.0, -1.0, 1.0, 5.0, 25.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0f, 0.0f, -15.0f);
}


static void
CleanUp(void)
{
    glDeleteShader(fragShader);
    glDeleteShader(vertShader);
    glDeleteProgram(program);
    glutDestroyWindow(win);
}


static void
Key(unsigned char key, int x, int y)
```

21

```
{
    const GLfloat step = 2.0;
    (void) x;
    (void) y;

    switch(key) {
    case 'a':
        Anim = !Anim;
        if (Anim)
            glutIdleFunc(Idle);
        else
            glutIdleFunc(NULL);
        break;
    case 'w':
        WireFrame = !WireFrame;
        break;
    case 'z':
        zRot += step;
        break;
    case 'Z':
        zRot -= step;
        break;
    case 27:
        CleanUp();
        exit(0);
        break;
    }
    glutPostRedisplay();
}


static void
SpecialKey(int key, int x, int y)
{
    const GLfloat step = 2.0;

    (void) x;
    (void) y;

    switch(key) {
    case GLUT_KEY_UP:
        xRot += step;
        break;
    case GLUT_KEY_DOWN:
        xRot -= step;
        break;
    case GLUT_KEY_LEFT:
        yRot -= step;
        break;
    case GLUT_KEY_RIGHT:
        yRot += step;
        break;
    }
    glutPostRedisplay();
}


static void
Init(void)
```

```c
{
    GLfloat HeightArray[20];
    GLint u, i;

    if (!ShadersSupported())
        exit(1);

    vertShader = CompileShaderText(GL_VERTEX_SHADER, VertShaderText);
    fragShader = CompileShaderText(GL_FRAGMENT_SHADER, FragShaderText);
    program = LinkShaders(vertShader, fragShader);

    glUseProgram(program);

    /* Setup the HeightArray[] uniform */
    for (i = 0; i < 20; i++)
        HeightArray[i] = i / 20.0;
    u = glGetUniformLocation(program, "HeightArray");
    glUniform1fv(u, 20, HeightArray);

    assert(glGetError() == 0);

    glClearColor(0.4f, 0.4f, 0.8f, 0.0f);
    glEnable(GL_DEPTH_TEST);
    glColor3f(1, 1, 1);
}


int
main(int argc, char *argv[])
{
    srand(time(NULL));
    float in = 0.0;
    int control = 0;


    FILE *f = fopen("results.txt", "r");
    fscanf(f,"%d",&LENGTH);
    fscanf(f,"%d",&control);
    assert(LENGTH==control);
    float arr[LENGTH*LENGTH];
    inm = &arr;
    for(int i=0; i<LENGTH*LENGTH; i++)
    {
        fscanf(f,"%f",&in);
        inm[i] = in;
        //inm[i] = rand()%GLmaxdiff;
        if (in < lowest)
        {
            lowest = in;
        }
        else if (in > highest)
        {
            highest = in;
        }
        //printf("%f\n", in);
    }
    fclose(f);
```

```
    /**
     * Start OpenGL stuff
     */

    glutInit(&argc, argv);
    glutInitWindowSize(500, 500);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    win = glutCreateWindow(argv[0]);
    glewInit();
    glutReshapeFunc(Reshape);
    glutKeyboardFunc(Key);
    glutSpecialFunc(SpecialKey);
    glutDisplayFunc(Redisplay);
    Init();
    if (Anim)
        glutIdleFunc(Idle);
    glutMainLoop();
    return 0;
}
```

Display support(glut_wrap.h)

```
#ifndef GLUT_WRAP_H
#define GLUT_WRAP_H

#ifdef HAVE_FREEGLUT
#  include <GL/freeglut.h>
#elif defined __APPLE__
#  include <GLUT/glut.h>
#else
#  include <GL/glut.h>
#endif

#ifndef GLAPIENTRY
#define GLAPIENTRY
#endif

#endif /* ! GLUT_WRAP_H */
```

Display support(shaderutil.c)

```
/**
 * Utilities for OpenGL shading language
 *
 * Brian Paul
 * 9 April 2008
 */


#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <GL/glew.h>
#include "glut_wrap.h"
#include "shaderutil.h"

/** time to compile previous shader */
static GLdouble CompileTime = 0.0;
```

```c
/** time to linke previous program */
static GLdouble LinkTime = 0.0;

PFNGLCREATESHADERPROC CreateShader = NULL;
PFNGLDELETESHADERPROC DeleteShader = NULL;
PFNGLSHADERSOURCEPROC ShaderSource = NULL;
PFNGLGETSHADERIVPROC GetShaderiv = NULL;
PFNGLGETSHADERINFOLOGPROC GetShaderInfoLog = NULL;
PFNGLCREATEPROGRAMPROC CreateProgram = NULL;
PFNGLDELETEPROGRAMPROC DeleteProgram = NULL;
PFNGLATTACHSHADERPROC AttachShader = NULL;
PFNGLLINKPROGRAMPROC LinkProgram = NULL;
PFNGLUSEPROGRAMPROC UseProgram = NULL;
PFNGLGETPROGRAMIVPROC GetProgramiv = NULL;
PFNGLGETPROGRAMINFOLOGPROC GetProgramInfoLog = NULL;
PFNGLVALIDATEPROGRAMARBPROC ValidateProgramARB = NULL;
PFNGLUNIFORM1IPROC Uniform1i = NULL;
PFNGLUNIFORM1FVPROC Uniform1fv = NULL;
PFNGLUNIFORM2FVPROC Uniform2fv = NULL;
PFNGLUNIFORM3FVPROC Uniform3fv = NULL;
PFNGLUNIFORM4FVPROC Uniform4fv = NULL;
PFNGLUNIFORMMATRIX4FVPROC UniformMatrix4fv = NULL;
PFNGLGETACTIVEATTRIBPROC GetActiveAttrib = NULL;
PFNGLGETATTRIBLOCATIONPROC GetAttribLocation = NULL;

static void GLAPIENTRY
fake_ValidateProgram(GLuint prog)
{
    (void) prog;
}

GLboolean
ShadersSupported(void)
{
    if (GLEW_VERSION_2_0) {
        CreateShader = glCreateShader;
        DeleteShader = glDeleteShader;
        ShaderSource = glShaderSource;
        GetShaderiv = glGetShaderiv;
        GetShaderInfoLog = glGetShaderInfoLog;
        CreateProgram = glCreateProgram;
        DeleteProgram = glDeleteProgram;
        AttachShader = glAttachShader;
        LinkProgram = glLinkProgram;
        UseProgram = glUseProgram;
        GetProgramiv = glGetProgramiv;
        GetProgramInfoLog = glGetProgramInfoLog;
        ValidateProgramARB = (GLEW_ARB_shader_objects)
            ? glValidateProgramARB : fake_ValidateProgram;
        Uniform1i = glUniform1i;
        Uniform1fv = glUniform1fv;
        Uniform2fv = glUniform2fv;
        Uniform3fv = glUniform3fv;
        Uniform4fv = glUniform4fv;
        UniformMatrix4fv = glUniformMatrix4fv;
        GetActiveAttrib = glGetActiveAttrib;
        GetAttribLocation = glGetAttribLocation;
        return GL_TRUE;
    }
```

```
    else if (GLEW_ARB_vertex_shader && GLEW_ARB_fragment_shader
            && GLEW_ARB_shader_objects) {
        fprintf(stderr, "Warning: Trying ARB GLSL instead of OpenGL 2.x.  This may not work.\n");
        CreateShader = glCreateShaderObjectARB;
        DeleteShader = glDeleteObjectARB;
        ShaderSource = glShaderSourceARB;
        GetShaderiv = glGetObjectParameterivARB;
        GetShaderInfoLog = glGetInfoLogARB;
        CreateProgram = glCreateProgramObjectARB;
        DeleteProgram = glDeleteObjectARB;
        AttachShader = glAttachObjectARB;
        LinkProgram = glLinkProgramARB;
        UseProgram = glUseProgramObjectARB;
        GetProgramiv = glGetObjectParameterivARB;
        GetProgramInfoLog = glGetInfoLogARB;
        ValidateProgramARB = glValidateProgramARB;
        Uniform1i = glUniform1iARB;
        Uniform1fv = glUniform1fvARB;
        Uniform2fv = glUniform2fvARB;
        Uniform3fv = glUniform3fvARB;
        Uniform4fv = glUniform4fvARB;
        UniformMatrix4fv = glUniformMatrix4fvARB;
        GetActiveAttrib = glGetActiveAttribARB;
        GetAttribLocation = glGetAttribLocationARB;
        return GL_TRUE;
    }
    fprintf(stderr, "Sorry, GLSL not supported with this OpenGL.\n");
    return GL_FALSE;
}


GLuint
CompileShaderText(GLenum shaderType, const char *text)
{
    GLuint shader;
    GLint stat;
    GLdouble t0, t1;

    shader = CreateShader(shaderType);
    ShaderSource(shader, 1, (const GLchar **) &text, NULL);

    t0 = glutGet(GLUT_ELAPSED_TIME) * 0.001;
    glCompileShader(shader);
    t1 = glutGet(GLUT_ELAPSED_TIME) * 0.001;

    CompileTime = t1 - t0;

    GetShaderiv(shader, GL_COMPILE_STATUS, &stat);
    if (!stat) {
        GLchar log[1000];
        GLsizei len;
        GetShaderInfoLog(shader, 1000, &len, log);
        fprintf(stderr, "Error: problem compiling shader: %s\n", log);
        exit(1);
    }
    else {
        /*printf("Shader compiled OK\n");*/
    }
    return shader;
```

```
}


/**
 * Read a shader from a file.
 */
GLuint
CompileShaderFile(GLenum shaderType, const char *filename)
{
    const int max = 100*1000;
    int n;
    char *buffer = (char*) malloc(max);
    GLuint shader;
    FILE *f;

    f = fopen(filename, "r");
    if (!f) {
        fprintf(stderr, "Unable to open shader file %s\n", filename);
        free(buffer);
        return 0;
    }

    n = fread(buffer, 1, max, f);
    /*printf("read %d bytes from shader file %s\n", n, filename);*/
    if (n > 0) {
        buffer[n] = 0;
        shader = CompileShaderText(shaderType, buffer);
    }
    else {
        fclose(f);
        free(buffer);
        return 0;
    }

    fclose(f);
    free(buffer);

    return shader;
}


GLuint
LinkShaders(GLuint vertShader, GLuint fragShader)
{
    return LinkShaders3(vertShader, 0, fragShader);
}


GLuint
LinkShaders3(GLuint vertShader, GLuint geomShader, GLuint fragShader)
{
    GLuint program = CreateProgram();
    GLdouble t0, t1;

    assert(vertShader || fragShader);

    if (vertShader)
        AttachShader(program, vertShader);
    if (geomShader)
```

```
        AttachShader ( program ,  geomShader ) ;
    if  ( fragShader )
        AttachShader ( program ,  fragShader ) ;

    t0 = glutGet (GLUT_ELAPSED_TIME) * 0.001;
    LinkProgram ( program ) ;
    t1 = glutGet (GLUT_ELAPSED_TIME) * 0.001;

    LinkTime = t1 − t0 ;

    /∗ check link ∗/
    {
        GLint stat ;
        GetProgramiv ( program ,  GL_LINK_STATUS,  &stat ) ;
        if  (! stat ) {
            GLchar log [1000];
            GLsizei len ;
            GetProgramInfoLog ( program ,  1000,  &len ,  log ) ;
            fprintf ( stderr ,  "Shader link error :\ n%s\n",  log ) ;
            return 0;
        }
    }

    return program ;
}


GLuint
LinkShaders3WithGeometryInfo (GLuint vertShader ,  GLuint geomShader ,  GLuint fragShader ,
                              GLint verticesOut ,  GLenum inputType ,  GLenum outputType )
{
    GLuint program = CreateProgram () ;
    GLdouble t0 ,  t1 ;

    assert ( vertShader  ||  fragShader ) ;

    if  ( vertShader )
        AttachShader ( program ,  vertShader ) ;
    if  ( geomShader ) {
        AttachShader ( program ,  geomShader ) ;
        glProgramParameteriARB ( program ,  GL_GEOMETRY_VERTICES_OUT_ARB,  verticesOut ) ;
        glProgramParameteriARB ( program ,  GL_GEOMETRY_INPUT_TYPE_ARB,  inputType ) ;
        glProgramParameteriARB ( program ,  GL_GEOMETRY_OUTPUT_TYPE_ARB,  outputType ) ;
    }
    if  ( fragShader )
        AttachShader ( program ,  fragShader ) ;

    t0 = glutGet (GLUT_ELAPSED_TIME) * 0.001;
    LinkProgram ( program ) ;
    t1 = glutGet (GLUT_ELAPSED_TIME) * 0.001;

    LinkTime = t1 − t0 ;

    /∗ check link ∗/
    {
        GLint stat ;
        GetProgramiv ( program ,  GL_LINK_STATUS,  &stat ) ;
        if  (! stat ) {
            GLchar log [1000];
```

```
        GLsizei len;
        GetProgramInfoLog(program, 1000, &len, log);
        fprintf(stderr, "Shader link error:\n%s\n", log);
        return 0;
    }
  }

  return program;
}


GLboolean
ValidateShaderProgram(GLuint program)
{
    GLint stat;
    ValidateProgramARB(program);
    GetProgramiv(program, GL_VALIDATE_STATUS, &stat);

    if (!stat) {
        GLchar log[1000];
        GLsizei len;
        GetProgramInfoLog(program, 1000, &len, log);
        fprintf(stderr, "Program validation error:\n%s\n", log);
        return 0;
    }

    return (GLboolean) stat;
}


GLdouble
GetShaderCompileTime(void)
{
    return CompileTime;
}


GLdouble
GetShaderLinkTime(void)
{
    return LinkTime;
}


void
SetUniformValues(GLuint program, struct uniform_info uniforms[])
{
    GLuint i;

    for (i = 0; uniforms[i].name; i++) {
        uniforms[i].location
            = glGetUniformLocation(program, uniforms[i].name);

        switch (uniforms[i].type) {
        case GL_INT:
        case GL_SAMPLER_1D:
        case GL_SAMPLER_2D:
        case GL_SAMPLER_3D:
        case GL_SAMPLER_CUBE:
```

```
        case GL_SAMPLER_2D_RECT_ARB:
        case GL_SAMPLER_1D_SHADOW:
        case GL_SAMPLER_2D_SHADOW:
        case GL_SAMPLER_1D_ARRAY:
        case GL_SAMPLER_2D_ARRAY:
        case GL_SAMPLER_1D_ARRAY_SHADOW:
        case GL_SAMPLER_2D_ARRAY_SHADOW:
            assert(uniforms[i].value[0] >= 0.0F);
            Uniform1i(uniforms[i].location,
                        (GLint) uniforms[i].value[0]);
            break;
        case GL_FLOAT:
            Uniform1fv(uniforms[i].location, 1, uniforms[i].value);
            break;
        case GL_FLOAT_VEC2:
            Uniform2fv(uniforms[i].location, 1, uniforms[i].value);
            break;
        case GL_FLOAT_VEC3:
            Uniform3fv(uniforms[i].location, 1, uniforms[i].value);
            break;
        case GL_FLOAT_VEC4:
            Uniform4fv(uniforms[i].location, 1, uniforms[i].value);
            break;
        case GL_FLOAT_MAT4:
            UniformMatrix4fv(uniforms[i].location, 1, GL_FALSE,
                                uniforms[i].value);
            break;
        default:
            if (strncmp(uniforms[i].name, "gl_", 3) == 0) {
                /* built-in uniform: ignore */
            }
            else {
                fprintf(stderr,
                        "Unexpected uniform data type in SetUniformValues\n");
                abort();
            }
        }
    }
}


/** Get list of uniforms used in the program */
GLuint
GetUniforms(GLuint program, struct uniform_info uniforms[])
{
    GLint n, max, i;

    GetProgramiv(program, GL_ACTIVE_UNIFORMS, &n);
    GetProgramiv(program, GL_ACTIVE_UNIFORM_MAX_LENGTH, &max);

    for (i = 0; i < n; i++) {
        GLint size, len;
        GLenum type;
        char name[100];

        glGetActiveUniform(program, i, 100, &len, &size, &type, name);

        uniforms[i].name = strdup(name);
        uniforms[i].size = size;
```

```c
        uniforms[i].type = type;
        uniforms[i].location = glGetUniformLocation(program, name);
    }

    uniforms[i].name = NULL; /* end of list */

    return n;
}


void
PrintUniforms(const struct uniform_info uniforms[])
{
    GLint i;

    printf("Uniforms:\n");

    for (i = 0; uniforms[i].name; i++) {
        printf("  %d: %s size=%d type=0x%x loc=%d value=%g, %g, %g, %g\n",
               i,
               uniforms[i].name,
               uniforms[i].size,
               uniforms[i].type,
               uniforms[i].location,
               uniforms[i].value[0],
               uniforms[i].value[1],
               uniforms[i].value[2],
               uniforms[i].value[3]);
    }
}


/** Get list of attribs used in the program */
GLuint
GetAttribs(GLuint program, struct attrib_info attribs[])
{
    GLint n, max, i;

    GetProgramiv(program, GL_ACTIVE_ATTRIBUTES, &n);
    GetProgramiv(program, GL_ACTIVE_ATTRIBUTE_MAX_LENGTH, &max);

    for (i = 0; i < n; i++) {
        GLint size, len;
        GLenum type;
        char name[100];

        GetActiveAttrib(program, i, 100, &len, &size, &type, name);

        attribs[i].name = strdup(name);
        attribs[i].size = size;
        attribs[i].type = type;
        attribs[i].location = GetAttribLocation(program, name);
    }

    attribs[i].name = NULL; /* end of list */

    return n;
}
```

```
void
PrintAttribs(const struct attrib_info attribs[])
{
    GLint i;

    printf("Attribs:\n");

    for (i = 0; attribs[i].name; i++) {
        printf("  %d: %s  size=%d  type=0x%x  loc=%d\n",
               i,
               attribs[i].name,
               attribs[i].size,
               attribs[i].type,
               attribs[i].location);
    }
}
```

Display support(shaderutil.h)

```
#ifndef SHADER_UTIL_H
#define SHADER_UTIL_H


#ifdef __cplusplus
extern "C" {
#endif


struct uniform_info
{
    const char *name;
    GLuint size;   /**< number of value[] elements: 1, 2, 3 or 4 */
    GLenum type;   /**< GL_FLOAT, GL_FLOAT_VEC4, GL_INT, GL_FLOAT_MAT4, etc */
    GLfloat value[16];
    GLint location;   /**< filled in by InitUniforms() */
};

#define END_OF_UNIFORMS    { NULL, 0, GL_NONE, { 0, 0, 0, 0 }, -1 }


struct attrib_info
{
    const char *name;
    GLuint size;   /**< number of value[] elements: 1, 2, 3 or 4 */
    GLenum type;   /**< GL_FLOAT, GL_FLOAT_VEC4, GL_INT, etc */
    GLint location;
};


extern GLboolean
ShadersSupported(void);

extern GLuint
CompileShaderText(GLenum shaderType, const char *text);

extern GLuint
CompileShaderFile(GLenum shaderType, const char *filename);

extern GLuint
```

```
LinkShaders(GLuint vertShader, GLuint fragShader);

extern GLuint
LinkShaders3(GLuint vertShader, GLuint geomShader, GLuint fragShader);

extern GLuint
LinkShaders3WithGeometryInfo(GLuint vertShader, GLuint geomShader, GLuint fragShader,
                             GLint verticesOut, GLenum inputType, GLenum outputType);

extern GLboolean
ValidateShaderProgram(GLuint program);

extern GLdouble
GetShaderCompileTime(void);

extern GLdouble
GetShaderLinkTime(void);

extern void
SetUniformValues(GLuint program, struct uniform_info uniforms[]);

extern GLuint
GetUniforms(GLuint program, struct uniform_info uniforms[]);

extern void
PrintUniforms(const struct uniform_info uniforms[]);

extern GLuint
GetAttribs(GLuint program, struct attrib_info attribs[]);

extern void
PrintAttribs(const struct attrib_info attribs[]);

/* These pointers are only valid after calling ShadersSupported.
 */
extern PFNGLCREATESHADERPROC CreateShader;
extern PFNGLDELETESHADERPROC DeleteShader;
extern PFNGLSHADERSOURCEPROC ShaderSource;
extern PFNGLGETSHADERIVPROC GetShaderiv;
extern PFNGLGETSHADERINFOLOGPROC GetShaderInfoLog;
extern PFNGLCREATEPROGRAMPROC CreateProgram;
extern PFNGLDELETEPROGRAMPROC DeleteProgram;
extern PFNGLATTACHSHADERPROC AttachShader;
extern PFNGLLINKPROGRAMPROC LinkProgram;
extern PFNGLUSEPROGRAMPROC UseProgram;
extern PFNGLGETPROGRAMIVPROC GetProgramiv;
extern PFNGLGETPROGRAMINFOLOGPROC GetProgramInfoLog;
extern PFNGLVALIDATEPROGRAMARBPROC ValidateProgramARB;
extern PFNGLUNIFORM1IPROC Uniform1i;
extern PFNGLUNIFORM1FVPROC Uniform1fv;
extern PFNGLUNIFORM2FVPROC Uniform2fv;
extern PFNGLUNIFORM3FVPROC Uniform3fv;
extern PFNGLUNIFORM4FVPROC Uniform4fv;
extern PFNGLGETACTIVEATTRIBPROC GetActiveAttrib;
extern PFNGLGETATTRIBLOCATIONPROC GetAttribLocation;

#ifdef __cplusplus
}
#endif
```

```
#endif /* SHADER_UTIL_H */
```