

# CS601: Software Development for Scientific Computing

Autumn 2021

Week7:

- Unstructured Grids, Tools (GDB)

# Last Week..

- Elliptic PDEs - theory and program representation
  - 4-point stencil and system of equations
  - Solving system of equations – Jacobi and Gauss-Seidel iteration
  - Program representation: classes Domain, GridFn, and Solution

# GDB

- GNU Debugger – A tool for inspecting your C/C++ programs
  - How to begin inspecting a program using gdb?
  - How to control the execution?
  - How to display, interpret, and alter memory contents of a program using gdb?
  - Misc – displaying stack frames, visualizing assembler code.

# GDB

- Compile your programs with `-g` option

```
hegden$gcc gdbdemo.c -o gdbdemo -g
hegden$
```

```
1 #include<stdio.h>
2 int foo(int a, int b)
3 {
4     int x = a + 1;
5     int y = b + 2;
6     int sum = x + y;
7
8     return x * y + sum;
9 }
10
11 int main()
12 {
13     int ret = foo(10, 20);
14     printf("value returned from foo: %d\n",ret);
15     return 0;
16 }
```

# GDB – Start Debug

- Start debug mode (gdb gdbdemo)
  - Note the executable on first line (not .c files)
  - Note the last line before (gdb) prompt:
    - if –g option is not used while compiling, you will see “(no debugging symbols found)”

```
[ecegrid-thin4:~/ECE264] hegden$gdb gdbdemo
GNU gdb (GDB) Red Hat Enterprise Linux (7.2-92.el6)
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/min/a/hegden/ECE264/gdbdemo...done.
(gdb)
```

# GDB – Set breakpoints

- Set breakpoints (b)

- At line 14
- Beginning of foo

```
1 #include<stdio.h>
2 int foo(int a, int b)
3 {
4     int x = a + 1;
5     int y = b + 2;
6     int sum = x + y;
7
8     return x * y + sum;
9 }
10
11 int main()
12 {
13     int ret = foo(10, 20);
14     printf("value returned from foo: %d\n",ret);
15     return 0;
16 }
```

```
(gdb) b gdbdemo.c:14
Breakpoint 1 at 0x400512: file gdbdemo.c, line 14.
(gdb) b foo
Breakpoint 2 at 0x4004ce: file gdbdemo.c, line 4.
(gdb) █
```

# GDB – Start execution

- Start execution (`r <command-line arguments>`)
  - Execution stops at the first breakpoint encountered

```
(gdb) r
Starting program: /home/min/a/hegden/ECE264/gdbdemo

Breakpoint 3, main () at gdbdemo.c:13
13      _      int ret = foo(10, 20);
```

- Continue execution (`c`)

```
(gdb) c
Continuing.

Program exited normally.
, " " ■
```

# GDB – Printing

- Printing variable values (p <variable\_name>)

```
Breakpoint 2, foo (a=10, b=20) at gdbdemo.c:4
4      int x = a + 1;
(gdb) n
5      int y = b + 2;
(gdb) p x
$3 = 11
```

- Printing addresses (p &<variable\_name>)

```
(gdb) p &x
$5 = (int *) 0x7fffffffcc4f4
```



# GDB – Manage breakpoints

- Display all breakpoints set (info b)

```
(gdb) info b
Num      Type          Disp Enb Address                What
1        breakpoint      keep y  0x0000000000400512 in main at gdbdemo.c:14
2        breakpoint      keep y  0x00000000004004ce in foo at gdbdemo.c:4
(gdb) █
```

- Delete a breakpoint (d <breakpoint num>)

```
(gdb) d 1
(gdb) info b
Num      Type          Disp Enb Address                What
2        breakpoint      keep y  0x00000000004004ce in foo at gdbdemo.c:4
(gdb) █
```

- Disable a breakpoint (disable <breakpoint num>)

```
(gdb) disable 2
(gdb) info b
Num      Type          Disp Enb Address                What
2        breakpoint      keep n  0x00000000004004ce in foo at gdbdemo.c:4
(gdb) █
```

- Enable breakpoint (enable <breakpoint num>)

```
(gdb) enable 2
(gdb) info b
Num      Type          Disp Enb Address                What
2        breakpoint      keep y  0x00000000004004ce in foo at gdbdemo.c:4
(gdb) █
```

# GDB – Step in

– Steps inside a function call (s)

```
Breakpoint 3, main () at gdbdemo.c:13
13         int ret = foo(10, 20);
(gdb) s
foo (a=10, b=20) at gdbdemo.c:4
4         _      int x = a + 1;
```

# GDB – Step out

– Jump to return address (finish)

```
(gdb) finish
Run till exit from #0  foo (a=10, b=20) at gdbdemo.c:4
0x000000000040050f in main () at gdbdemo.c:13
13      int ret = foo(10, 20);
Value returned is $2 = 275
```

# GDB – Memory dump

– Printing memory content (x/nfu <address>)

- n = repetition (number of bytes to display)
- f = format ('x' – hexadecimal, 'd'-decimal, etc.)
- u = unit ('b' – byte, 'h' – halfword/2 bytes, 'w' – word/4 bytes, 'g' – giga word/8 bytes)
- E.g. x/16xb 0x7fffffff500 (display the values of 16 bytes stored from starting address)

```
(gdb) x/16xb 0x7fffffff500
0x7fffffff500: 0x20    0xc5    0xff    0xff    0xff    0x7f    0x00    0x00
0x7fffffff508: 0x0f    0x05    0x40    0x00    0x00    0x00    0x00    0x00
```

# GDB – Printing addresses

## – Registers (\$rsp, \$rbp)

- Note that we use the 'x' command and not the 'p' command.

```
(gdb) x $rsp
0x7fffffffcc500: 0x20
(gdb) x $rbp
0x7fffffffcc500: 0x20
```

# GDB – Altering memory content

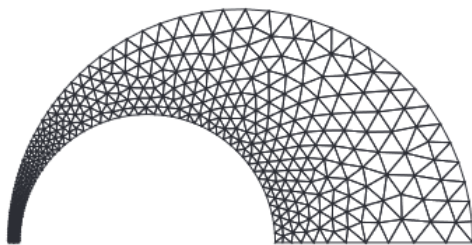
- Set command (set variable <name> = value)

```
(gdb) n
6          int sum = x + y;
(gdb) p x
$7 = 11
(gdb) p y
$8 = 22
(gdb) set variable y = 0
(gdb) n
8          return x * y + sum;
(gdb) p sum
$9 = 11
```

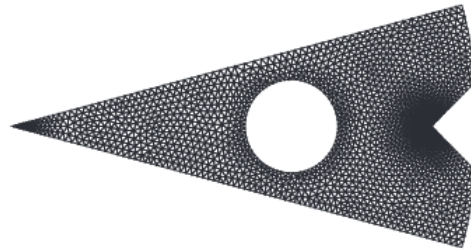
- Set command (set \*(<type \*>addr) = value)

# Unstructured Grids

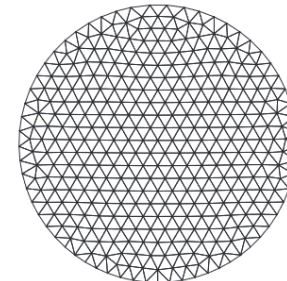
- Motivation:
  - E.g. reduce the noise because of air flowing through a duct, Maintain uniformity in flow (no pressure drop)



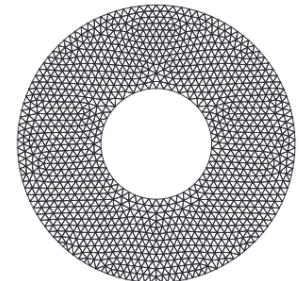
(e) Geometric Adaptivity



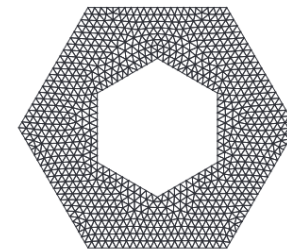
(f) Pie with hole



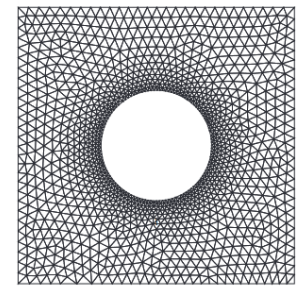
(a) Unit circle



(b) Unit circle with hole



(c) Square with hole (uniform)



(d) Square with hole (refined at hole)

<https://www.math.uci.edu/~chenlong/Papers/Chen.L%3BHolst.M2010.pdf>

- Handle complex geometries
- Refine at region of interest

# Unstructured Grids - Problem

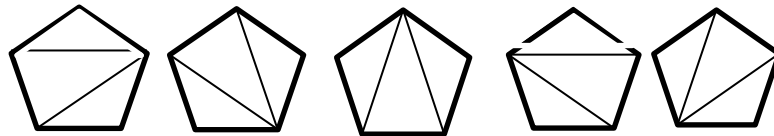
- Discretize the domain into an optimal set of triangles (or tetrahedra / simplex)
- Areas: Mesh Generation and Mesh Optimization

*Iterate*

1. Fix the location of vertices and optimize **OR**
2. Fix the vertices' connectivity and optimize (determine optimal placement of vertices)

( Optimize based on:

- Sum of edge weights



- Harmonic energy, Distortion energy )



# Unstructured Grids – Program Representation (partial)

- Option 1:

```
Point points[n]; // represents coordinates  
int triangles[m][3]; // represents triangles
```

- Option 2:

```
double x[n], y[n]; // represents coordinates (2D)  
int triangles[m][3]; // represents triangles
```

- Option 3:  $G(V, E)$

```
double x[n], y[n]; // represents vertices/nodes (2D)  
int edges[m][2]; // represents edges
```

# Unstructured Grids - Challenges

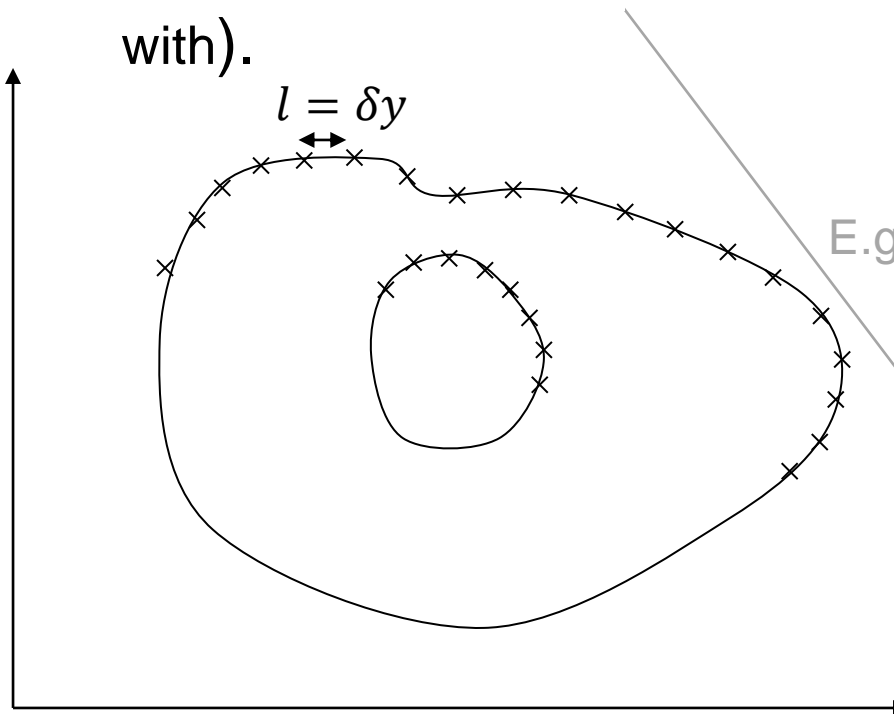
- Placing the points
  - What constitutes inside and what is outside the domain? *Place points only in the interior.*
  - What should be the distance between points?
  - How many points should be there?
- Connecting the points
  - What is the best way to create tiles once the points are placed?

# Unstructured Grids - Approaches

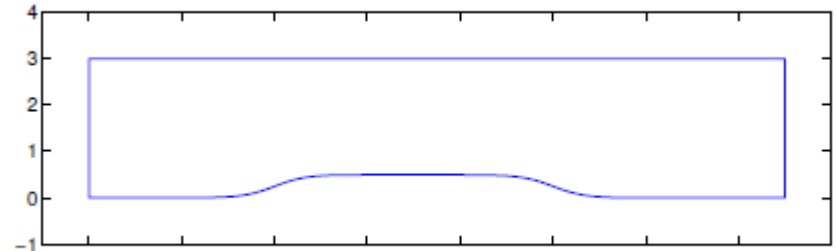
- Delaunay Triangulation is the most commonly used unstructured triangulation method
  - Advantage: can automatically give a ‘better’ triangulation (e.g. w.r.t aspect ratio)
  - Disadvantage: suitable for convex domain
- Advancing Front Method is another method
  - Advantage: suitable for concave domain
  - Disadvantage: No prioritization of triangulation

# Grid Generation - Example

- Draw *equidistant* points on the boundary (note: you have accurate description of the bounding curves of the domain to begin with).



E.g

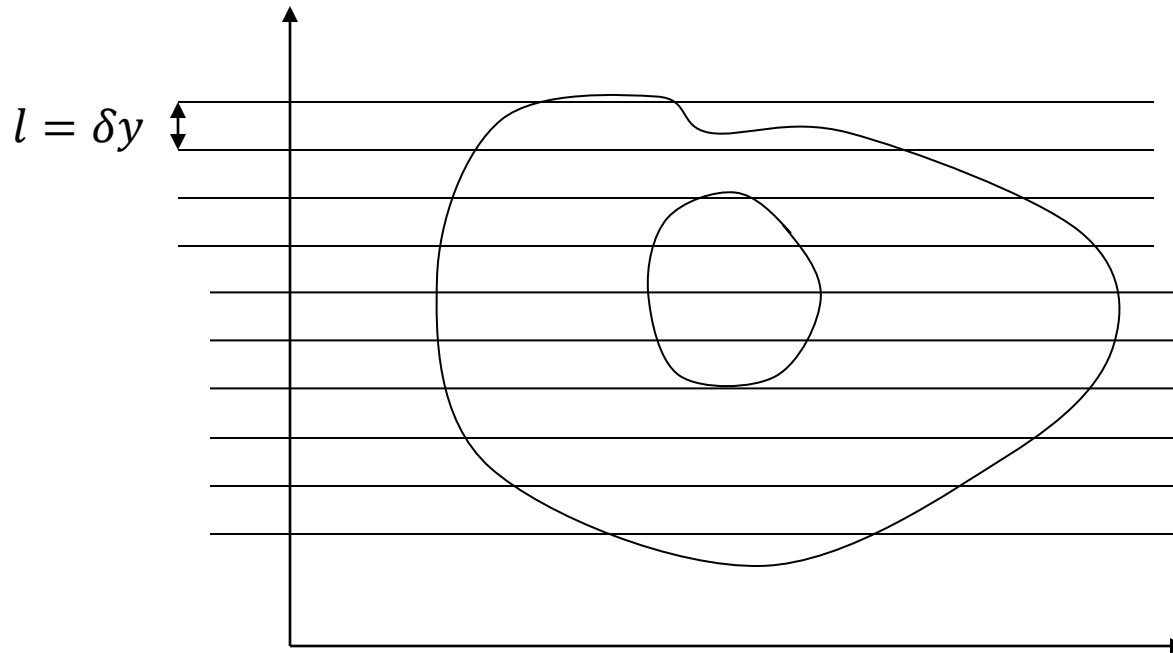


$$f(x) = \begin{cases} \frac{1}{2} \frac{1}{1 + \exp(-3(x+6))}, & x \in [-10, -3] \\ \frac{1}{2} \frac{1}{1 + \exp(3x)}, & x \in [-3, 5] \end{cases}$$

- What should be the distance?

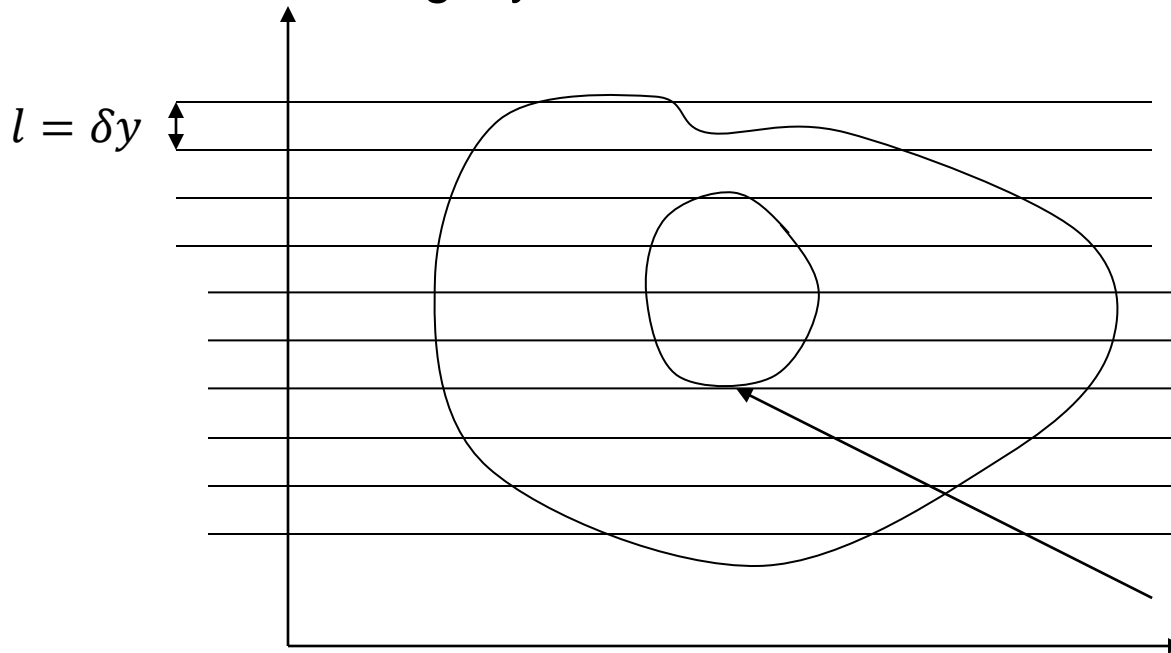
# Grid Generation - Example

- Pick any axis and slice the domain with equidistant (perp. to axis) lines. Now you may have odd or even number of intersecting points with the domain.



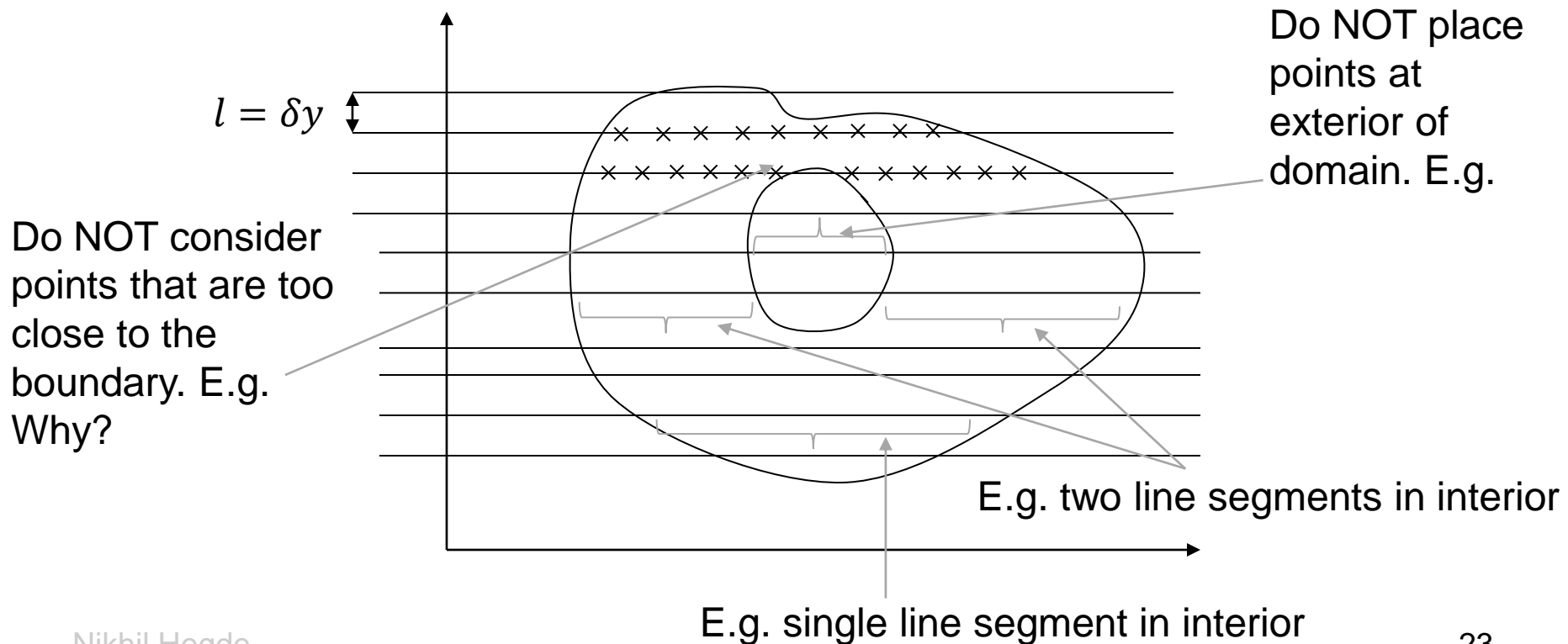
# Grid Generation - Example

- (Re)consider lines with odd number of intersecting points. Why?
  - Move the lines slightly around

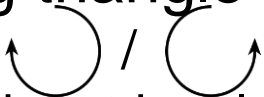


# Grid Generation - Example

- Pick line segments at the interior of the domain (you may have one or two in the example) and place equidistant ( $l$ ) points to fill up the interior with points



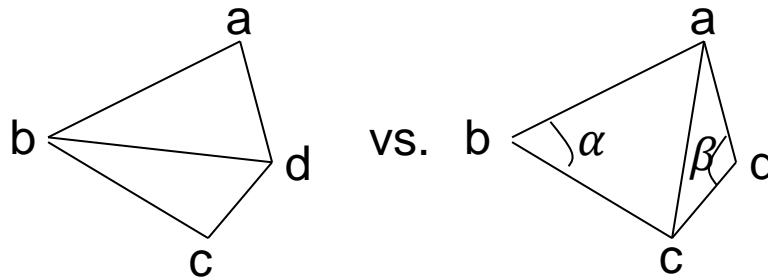
# Delaunay Triangulation

- You are given points/nodes on the boundaries and interior. Start with a super triangle
- Pick a node and find the enclosing triangle
  - How to find? Move along the triangle  and find if the point lies on RHS / LHS. Point is inside a triangle if for all edges, the point lies on RHS / LHS
- Split the enclosing triangle by connecting the picked node with the nodes of the enclosing triangle
- (Re) triangulate partial domain: make adjustments based on the best possible current scenario (with existing domain and existing nodes. You don't know what points are going to come into the domain in future).



# Delaunay Triangulation (Re-triangulate)

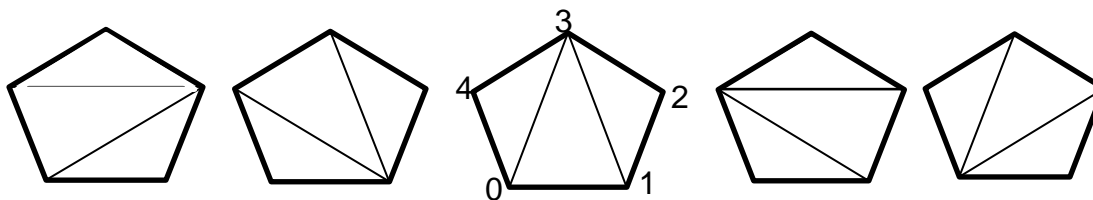
- Run e.g. Minimum Weight Triangulation on the newly triangulated polygon
  - Consider all quadrilaterals formed with existing triangles and the new point picked.
  - There are two ways of triangulating a quadrilateral. Pick the one that is better



- If  $(\sin(\alpha + \beta) < 0)$  then choose the other option
  - Equivalently, draw a circle around the triangle abc (circumcircle). If the fourth point, d, lies inside then choose the other option.

# Minimum Weight Triangulation

- Type of divide-and-conquer with two properties:
  - Optimal substructure and repeated sub-problems.



## Minimum Weight Triangulation Problem

Objective: Triangulate a polygon such that edges do not intersect AND sum of edge lengths is minimized

$$C(i, j) = \begin{cases} \min( C(i, j), \min_{i < k < j} C(i, k) + C(k, j) + W(i, k, j) ) & j \leq i + 1 \\ 0 & \text{Given } W(i, j, k) \end{cases}$$

# Minimum Weight Triangulation

- Pseudocode and call tree of triangulating a pentagon (vertices named 0 to 4)



Minimum Weight Triangulation Problem

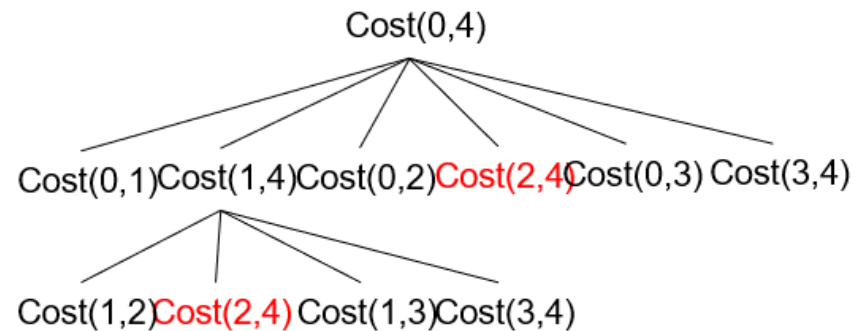
Objective: Triangulate a polygon such that edges do not intersect AND sum of edge lengths is minimized

$$C(i, j) = \begin{cases} \min_{i < k < j} (C(i, k) + C(k, j) + W(i, k, j)) & j \leq i + 1 \\ 0 & \text{Given } W(i, j, k) \end{cases}$$

```

Cost(i, j){
  if j < i+2 then return 0;
  curMin ← INFINITY;
  for k ← i+1 to j-1 do
    res ← Cost(i, k) + Cost(k, j) + Weight(i, k, j)
    if res < curMin then curMin ← res;
  return curMin;
}
Weight(i, j, k){
  // return sum of Euclidian distances between
  // (i, j), (j, k), and (k, i)
}

main(){
  Cost(0, 4);
}
    
```



# Minimum Weight Triangulation

- Iterative formulation (note the 2D array representing the matrix to be computed)



**Minimum Weight Triangulation Problem**

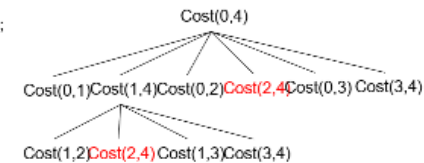
Objective: Triangulate a polygon such that edges do not intersect AND sum of edge lengths is minimized

$$C(i, j) = \begin{cases} \min_{i < k < j} (C(i, k) + C(k, j) + W(i, k, j)) & j \leq i + 1 \\ 0 & \text{Given } W(i, j, k) \end{cases}$$

```

Cost(i, j){
  If j < i+2 then return 0;
  curMin ← INFINITY;
  for k ← i+1 to j-1 do
    res ← Cost(i,k) + Cost(k,j) + Weight(i,k,j)
    If res < curMin then curMin ← res;
  return curMin;
}
Weight(i,j,k){
  // return sum of Euclidian distances between
  // (i,j), (j,k), and (k,i)
}

main(){
  Cost(0,4);
}
    
```



```

1 Cost(n){
2   table[n][n]; //n is number of vertices
3   for g ← 1 to n-1 do
4     for i ← 0 to n-g do
5       j ← i + g;
6       table[i][j] ← INFINITY;
7       for k ← i+1 to j-1 do
8         res ← table[i][k] + table[k][j] +
Weight(i,k,j)
9         if res < table[i][j] then
10          table[i][j] ← res;
}
    
```

**Iterative formulation**

# Minimum Weight Triangulation

- The 2D array is used to compute only the upper triangular matrix. Cost of polygon (0,1,2,3) is shown.

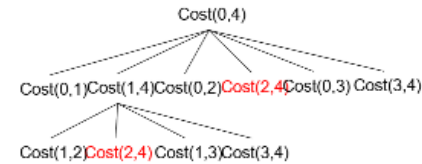


Minimum Weight Triangulation Problem

Objective: Triangulate a polygon such that edges do not intersect AND sum of edge lengths is minimized

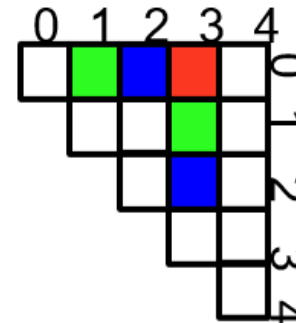
$$C(i, j) = \begin{cases} \min_{i < k < j} (C(i, j), \min_{i < k < j} C(i, k) + C(k, j) + W(i, k, j)) \\ 0 & j \leq i + 1 \\ \text{Given } W(i, j, k) \end{cases}$$

```
Cost(i, j){
  If j < i+2 then return 0;
  curMin ← INFINITY;
  for k ← i+1 to j-1 do
    res ← Cost(i, k) + Cost(k, j) + Weight(i, k, j)
    If res < curMin then curMin ← res;
  return curMin;
}
main(){
  Cost(0, 4);
}
Weight(i, j, k){
  // return sum of Euclidian distances between
  // (i, j), (j, k), and (k, i)
}
```



```
1 Cost(n){
2   table[n][n]; //n is number of vertices
3   for g ← 1 to n-1 do
4     for i ← 0 to n-g do
5       j ← i + g;
6       table[i][j] ← INFINITY;
7       for k ← i+1 to j-1 do
8         res ← table[i][k] + table[k][j] +
Weight(i, k, j)
9         if res < table[i][j] then
10          table[i][j] ← res;
}
```

Iterative formulation



$$\text{Cost}(0,3) = \min(\text{Cost}(0,1) + \text{Cost}(1,3) + \triangle^{\text{le}}(0,1,3), \text{Cost}(0,2) + \text{Cost}(2,3) + \triangle^{\text{le}}(0,2,3))$$

# Further reading/viewing

- <https://www.youtube.com/watch?v=IPcBX4BBW9U>
- <https://www.youtube.com/watch?v=tWf1z9i-Org>
- <https://www.math.uci.edu/~chenlong/Papers/Chen.L%3BHolst.M2010.pdf>

# Midsem Review

- Computational thinking
  - Examples and analogy, 2As
- Non-integral real numbers as floating-point values
  - Scientific notation, precision, unit roundoff, epsilon.
- PDEs and Motifs
  - PDE Categorization, structured grids and stencil codes, unstructured grids (Delaunay triangulation)
- C++ programming
  - OO features, template programming, misc programming.
- Misc topics: System architecture, locality, software elements
- Tools
  - Git, Make, GDB

# Resources

- Refer to GDB cheat sheet:

<https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf>

- Makefile cheat sheet:

<https://bytes.usc.edu/cs104/wiki/makefile/>

- Git cheat sheet:

Thanks to GitHub education! (collect a hardcopy from office)