Solving Navier Stokes with 2D Spectral Element Method on a Block Element Mesh

Adil Ansari

Contents

Introduction

In the field of Computational Fluid Dynamics there are variety of methods to solve fluid simulation. While methods exist that focus on steady state, transient simulations give us an insight into the dynamic behavior of fluids. Such simulations can be simulated implicitly and explicitly in time. The benefit of implicit scheme is higher time steps even though it takes significantly longer per time step iteration. Spectral Element method are a class of method that take individual elements of a mesh and piece together an algebraic relationship to all other nodes in the simulation with individual element having spectral relationship with nodes within the same element. This enables us to solve PDEs with spectral accuracy while still retaining the sparsity of our system which is important for solving quicker. It also allows for have non-square or non-cube domains.

Project Problem

The Equation to be simulated in this project are as follows

	$$\frac{\partial \vec{u}_i}{\partial t} + u\frac{\partial \vec{u}_i}{\partial x} + v\frac{\partial \vec{u}_i}{\partial y} - \nabla_i p = \nu \Delta \vec{u}_i + f(T)$$	(1)
	$$\sum_{i=0}^{1} \nabla_i \vec{u}_i = 0$$	(2)
	$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} = \alpha \Delta T$$	(3)

Note f(T) is merely a forcing function to simulate a rough behavior of thermal currents and they lead to less dense rising fluids.

Mesh

Block Mesh and Grid Correction

At first, we generate a block mesh on basis of pixels which with previous knowledge of element density.

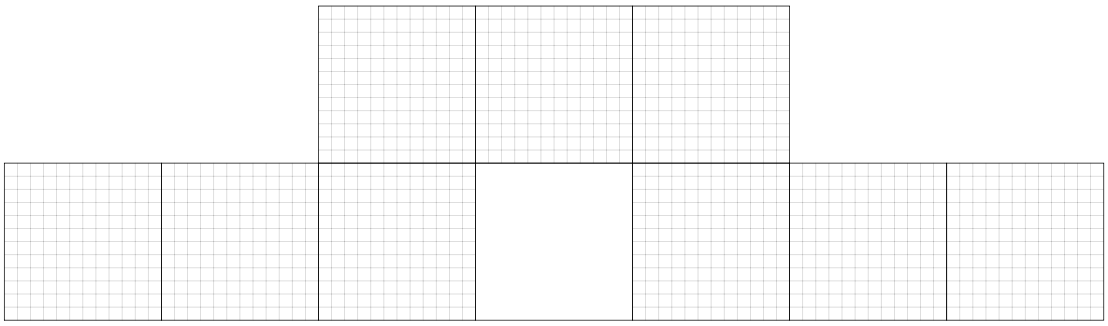


Figure 1: Block Mesh

Each l_x by l_y block contains E^2 Elements which contain N^2 Chebyshev Cells. Once we have generated the block mesh we need to distort to create 2D Chebyshev elements as such.

$$x^* = x - \frac{c\left(\frac{Ex}{l_x}\right)}{2El_x}, c(x) = cos\left(\pi\ (x\ mod\ 1)\right) + (2\ (x\ mod\ 1) - 1) \quad (4)$$

On implementation of this we get the following result.

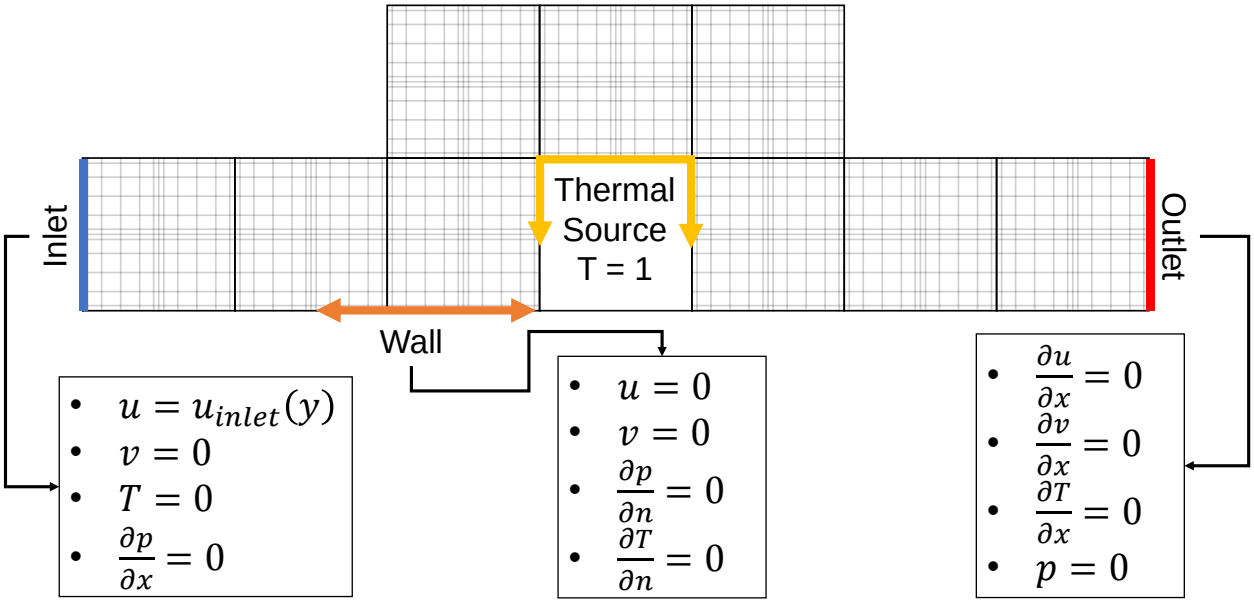


Figure 2: Block Element Mesh with Boundary Conditions with N = 6 and E = 2

The resultant mesh consists of blocks that contain elements of 2D Chebyshev grid distribution which is mapped on a connectivity matrix. We can define Inlet and Outlet conditions at leftmost and right most parts and everything else can be considered as walls. The central block that is surrounded by the bend is a constant temperature 'heated block' and hence Dirichlet boundary conditions are applied at the wall nodes on the temperature field. See figure 2 for all boundary conditions.

Numerical Method

Differentiation Matrix

Let D_N be the Chebyshev differentiation matrix of size N+1 by N+1 which can be computed as essentially $F^{-1}AF$ where F takes us from Chebyshev space to real space and A is differentiation matrix in Chebyshev space resulting from the recursive relationship of derivatives. To solve any PDE, we need to ascertain a global differentiation matrix with which we can compute a 2D Chebyshev differentiation matrix.

$$D_x^e = -I_n \otimes D_N, D_y^e = -D_N \otimes I_n \quad (5)$$

This 2D Chebyshev differentiation matrix can be accumulated into a global differentiation matrix while dividing out the overlapping nodes. Note L is a location matrix. Here $\otimes$ stands for Kronecker delta tensor product.

$$D_i = \sum_{e=1}^{N_e} L_e \frac{2D_i^e E}{l_i} L_e^T \oslash \sum_{e=1}^{N_e} L_e J_{N,1} \quad (6)$$

Here ⊘ stands for elementwise division. The global differentiation matrix can be used to compute a Laplacian Matrix.

$$D_l = D_x^2 + D_y^2 \quad (7)$$

Pressure Correction Step

Let L be location matrix, we can compute the Laplacian matrix for internal points and have differentiation matrices for the Neumann boundary condition.

$$B = L_{\Omega^e} D_l + L_{\Gamma^{\text{inlet,vertical walls}}} D_x + L_{\Gamma^{\text{horitzontal walls}}} D_y + L_{\Gamma^{\text{outlet}}} \quad (8)$$

Similarly compute the divergence on boundary points

$$d = \frac{L_{\Omega^e}}{\Delta t} \sum_{i=0}^{1} \nabla_i \vec{u}_i^t \quad (9)$$

Once that is done we compute pressure using sparse matrix LU decomposition or ILU preconditioner on some iterative solver.

$$p^{t+\Delta t} = B^{-1} d \quad (10)$$

Advection Step

Similarly, velocity Advection Step can be computed as:

$$A' = \frac{I}{\Delta t} - \frac{\nu D_l}{2} + \frac{U D_x}{2} + \frac{V D_y}{2} \quad (11)$$

$$r' = \left(\frac{I}{\Delta t} + \frac{\nu D_l}{2} - \frac{U D_x}{2} - \frac{V D_y}{2}\right) \vec{u}_i{}^t - \nabla_i p^{t+\Delta t} \quad (12)$$

$$A = L_{\Omega^e} A' + L_{\Gamma^{\text{walls,inlet}}} + L_{\Gamma^{\text{outlet}}} D_x \quad (13)$$

$$r = L_{\Omega^e} r' + L_{\Gamma^{\text{inlet}}} \vec{u}_{inlet}^t(y) \quad (14)$$

$$\vec{u}_i{}^{t+\Delta t} = A^{-1} r \quad (15)$$

where ν is kinematic viscosity, t is time and Δt is the time step. U and V are matrices with diagonal entries of horizontal and vertical velocity components at every point.

Matrix Solver

The solver used here is MATLAB direct sparse LU decomposition solver. This comes at a cost of high memory requirements and hence iterative solver with ILU preconditioner could have been used but in the interest of speed and interest of fast solving, the direct solver was used with the addition of Sparse reverse Cuthill-McKee ordering to speed up solving process.

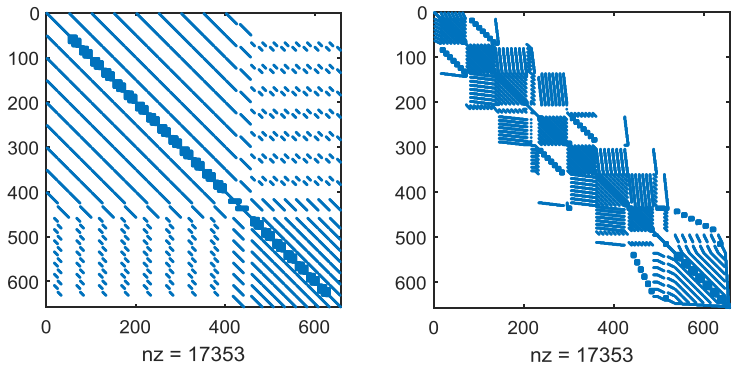


Figure 3: On the left is the original matrix and on the right we have the same matrix reordered.

Results

The inlet velocity profile is $u(y) = 1 - (2y - 1)^8$. The outlet pressure is set to 0 and every other field is set to zero gradient in the x-direction at the outlet. The value of ν is 0.1 and the value of α for Temperature diffusion is 0.1. Additionally there is forcing function on the v velocity to represent forced convection using boussinesq approximation.

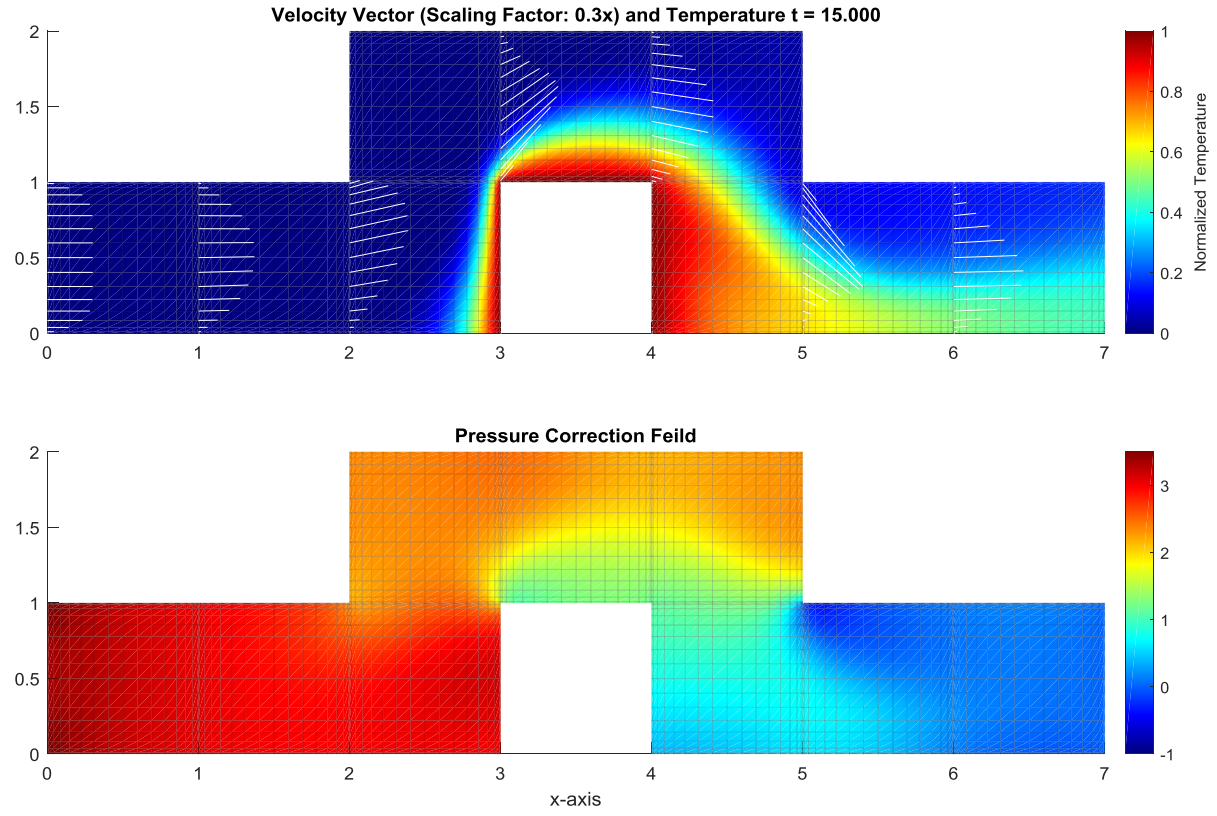


Figure 4: Solution of Flow around a bend with 16 Chebyshev nodes per block with v=0.1 and α=0.1.

The solution is steady. We can see sharp pressure drops near corners which makes sense they are resist flow inertially and also have the highest shear at those location.

Convergence

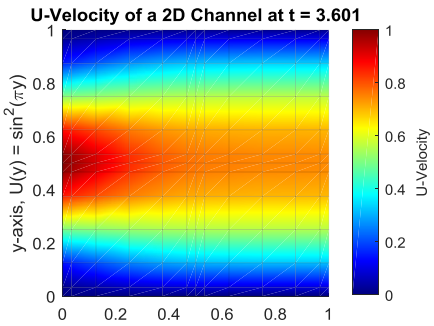


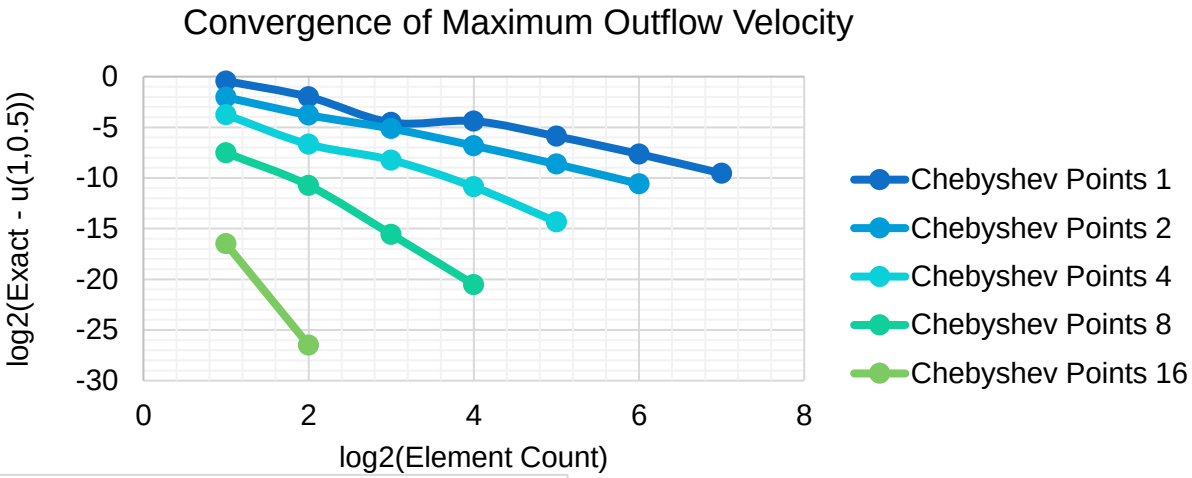
Figure 5: Mesh Resolution: E=2, N = 6 with v=1

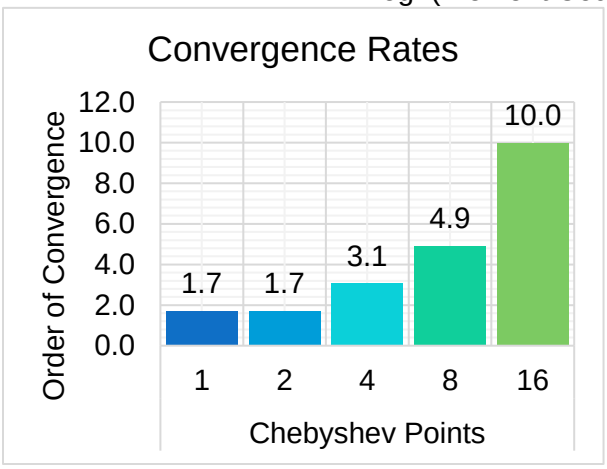
For Convergence we simplify our problem to a single block and set the inlet boundary condition to a sin squared function as shown on the left and we also set this as the initial condition as well. Temperature field is not part of this simulation. Since channel flow tend towards parabolic flow, ideally, we would go 0.667m/s (ascertained by mass conservation equation of parabolic profile) if the outlet was far enough from the inlet. But in this configuration, it has a different value of 0.74102m/s due to non-linearities. Hence, we try different sets of element count and

Chebyshev nodes count which are powers of 2 and after temporal convergence we get the following values of u(1,0.5):

Element Count (E) \ Chebyshev Resolution (N)	1	2	4	8	16	32
1	0.000000	0.984283	0.667183	0.735575	0.741009	**0.7410197**
2	0.992110	0.668939	0.731279	0.740434	0.741020	**EXACT**
4	0.697730	0.712346	0.737696	0.740999		
8	0.693301	0.732156	0.740488	0.741019		
16	0.724154	0.738511	0.740972			
32	0.736009	0.740365				
64	0.739683					

The exact solution was computed using 32 Chebyshev nodes on both directions as it is assumed at that level we are practically converged with error less than zero precision. The error





is converging at linear rates with element counts but the rate of convergence is proportional to the number of Chebyshev cells. So, in the end we have weigh the pros and cons of going with higher Chebyshev nodes as they have superior accuracy but are also RAM intensive processes when being solved.

Other Trials and Solutions

Here we compare high peclet number solution with low and high peclet number.

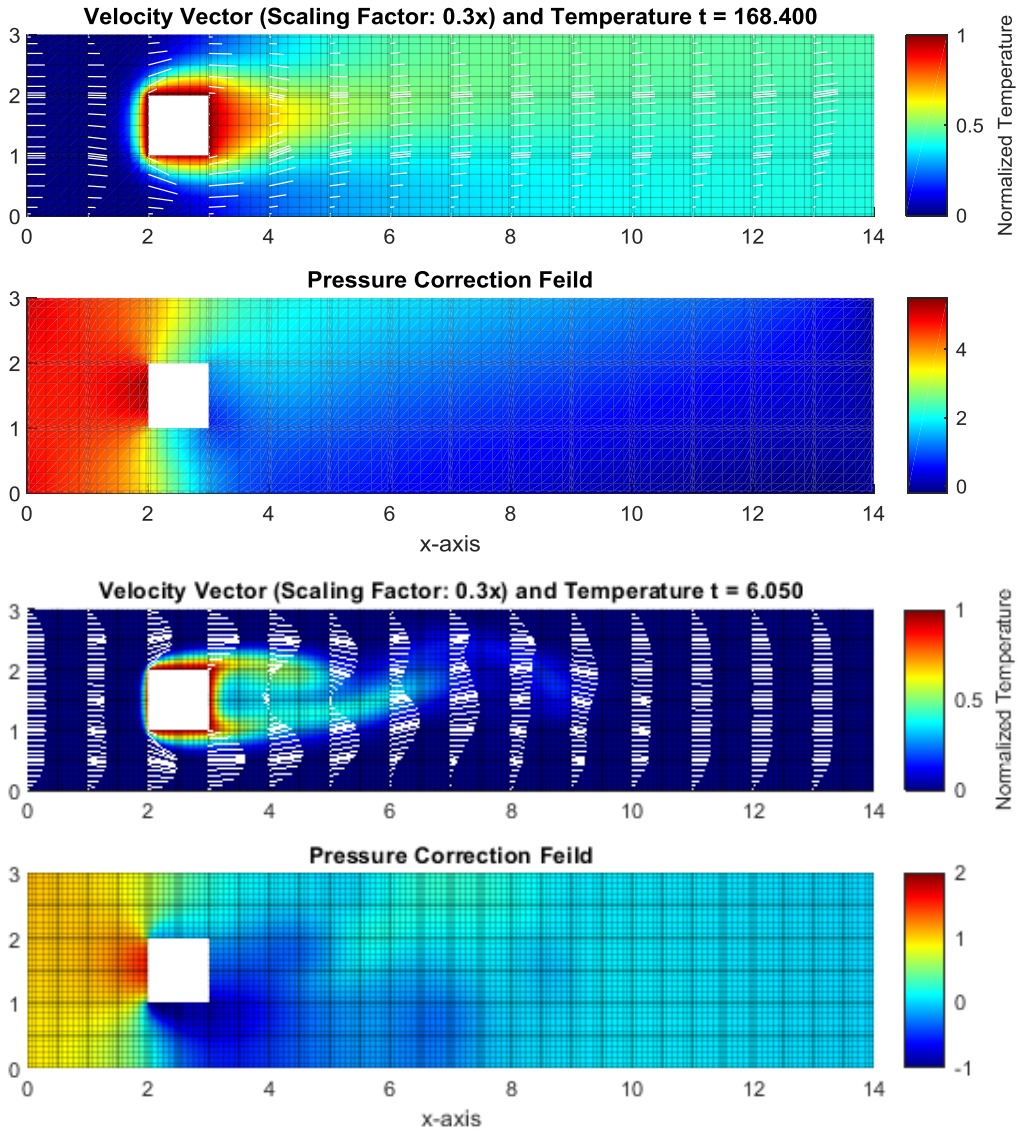


Figure 6: High Peclet Number with v=0.1 and α=0.1 on the top and v=0.01 and α=0.01 on the bottom.

With high diffusion rate of transport terms, we get a stable solution. However, when we decrease the viscosity we start to see transient structures form. These are not exactly von-Karman vortex street but rather a forced thermal current which leads to oscillatory behavior despite being well into the laminar regime as the Reynolds number is only 100 on the bottom and 10 on the top. Note the element resolution was doubled for lower viscosity case but despite that numerical error was noticed near bottom left corner of the cube in the form of a spurious oscillation vacillating the pressure field dramatically over time as everything is coupled. In the high viscosity case this doesn't happen but there is asymmetry in the pressure across y = 1.5 line as the thermal currents are pushing the fluids upwards.

As a joke I wanted to see what would happen if created a block airfoil and see what I get and their results were disappointing but made sense as the viscosity is too high to have any reasonable resemblance of real airfoil. This is clearly and illegal way of simulating an airfoil. But interestingly enough, one can use finite element methods to solve mazes albeit the most useless functionality of spectral element methods.

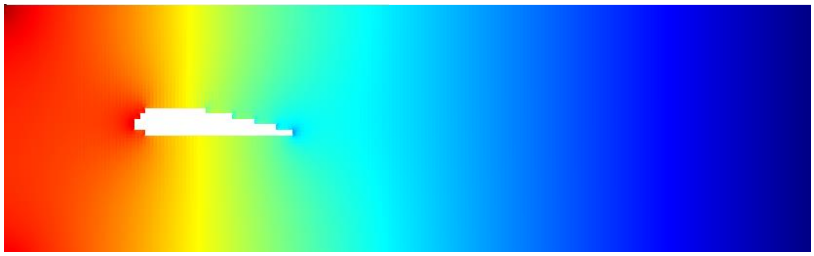
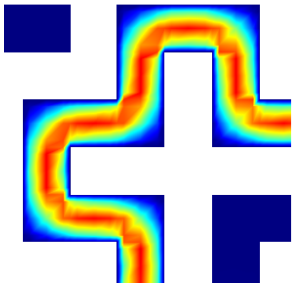


Figure 7; Maze solved using Stokes Flow on Chebyshev Element Grid

Conclusion

In conclusion Chebyshev spectral element methods are a powerful set of methods that can be used to simulate fluid dynamics and other transient PDEs. We can achieve spectral convergence while not being restricted on typical spectral domains. As with any other method there is tradeoff between computational resources and accuracy. Things I wanted to try in this project was application of gauss-lobatto quadrature as that would have enabled non-block meshes. Also, I wanted use anti-aliasing techniques to interpolate higher resolution images that produce better and 'cleaner' images than this bilinear interpolated patch on the right.



References

Lloyd N. Trefethen. 2000. Spectral Methods in Matlab. Soc. for Industrial and Applied Math., Philadelphia, PA, USA.

Jay Oswald. 2017. "http://compmech.lab.asu.edu"

Appendix

Codes

Chebyshev Differentiation Matrix Generator

```
function [FAF,A,F,B] = ChebydiffMatrix(N)
if nargin == 0
```

```
    N = 8;
end
L = [speye(N+1);flip(speye(N,N+1))];
L = L(1:end-1,:);
W = fft(eye(2*N));
WI=conj(W)/(2*N);
Fs = sparse(1:N+1,1:N+1,[1 2*ones(1,N-1) 1],N+1,2*N);
F = Fs*WI*L;
c = @(m) double(2.0-(m~=0));
m = uint16(1:N+1)';
p = uint16(1:N+1);
A = (sparse(logical(mod(m+p,2)) & logical((m<p))).*(2*(double(p)-1))./c(m-
1);
FAF = F\(A*F); %This matrix gives chebyshev derivative directly
band=[0 0 1./(2*(1:N+2))]'; k=1:N+1;
B=spdiags([band(k+2) -band(k)],[-1,1],N+1,N+1);
B(2,1)=1;
B = B(2:N+1,1:N);
```

Mesh Generator

```
function
[Dx,Dy,D2,D,num_nodes,X,conn,plotconn,exit,entry,hwall,vwall,interior] =
meshgen(E,N,lx,ly,map)
if nargin == 0
    E = 1;
    N = 4;
    lx = 1;
    ly = 1;
    map = [1 0;1 0;1 1 ; 0 1;1 1;1 0 ;1 0];
end
map2 = zeros(E*N*size(map,1)+1,E*N*size(map,2)+1,'int32');
fc = @(x) cos(pi*mod(x,1))+(2*mod(x,1)-1);
colormap(jet(512))

% map points from map
for j = 1:size(map,2)
    for i = 1:size(map,1)
        if map(i,j)
            map2((E*N)*(i-1)+(1:E*N+1),(E*N)*(j-1)+(1:E*N+1)) = 1;
        end
    end
end
pts = 1:sum(map2(:));
map2(map2==1) = pts;
num_nodes = pts(end);
conn = zeros((N+1)^2,sum(map(:))*E^2);
count = 0;
%Establish Boundary Points
vw = []; vc = 0;
hw = []; hc = 0;
for j = 2:size(map2,2)-1
    for i = 2:size(map2,1)-1
        if map2(i,j)
            if (~~map2(i+1,j) - ~~map2(i,j)) || (~~map2(i,j) - ~~map2(i-1,j))
                vc = vc + 1;
                vw(vc) = map2(i,j);
```

```
            elseif (~~map2(i,j+1) - ~~map2(i,j)) || (~~map2(i,j) -
~~map2(i,j-1))
                hc = hc + 1;
                hw(hc) = map2(i,j);
            elseif (~~map2(i,j+1) - ~~map2(i,j)) || (~~map2(i,j) -
~~map2(i,j-1)) ...
                    || (~~map2(i+1,j) - ~~map2(i,j)) || (~~map2(i,j) -
~~map2(i-1,j)) ...
                    || (~~map2(i+1,j+1) - ~~map2(i-1,j-1)) || (~~map2(i+1,j-
1) - ~~map2(i-1,j+1))
                hc = hc + 1;
                hw(hc) = map2(i,j);
            end
        end
    end
end
hw = [hw map2(map2(:,1)~=0,1)' map2(map2(:,end)~=0,end)'];
pe = map2(1,map2(1,:)~=0)';
hw = [hw pe([1 end])'];
pe = pe(2:end-1);
px = map2(end,map2(end,:)~=0)';
hw = [hw px([1 end])'];
px = px(2:end-1);
for j = 1:E*size(map,2)
    for i = 1:E*size(map,1)
        if map(ceil(i/E),ceil(j/E))
            temp = map2((N)*(i-1)+(1:N+1),(N)*(j-1)+(1:N+1));
            count = count + 1;
            conn(:,count) = temp(:);
        end
    end
end
count = 0;
X = zeros(2,num_nodes);
%Extablish point location
for j = 1:size(map2,2)
    for i = 1:size(map2,1)
        if map2(i,j)
            count = count + 1;
            X(1,count) = (i-1)/(E*N)*lx;
            X(2,count) = (j-1)/(E*N)*ly;
        end
    end
end
X(1,:) = X(1,:)-fc(X(1,:)*E/lx)/(2*E/lx);
X(2,:) = X(2,:)-fc(X(2,:)*E/ly)/(2*E/ly); %Fix Vertices
i = 1:size(map2,1)-1;
j = 1:size(map2,2)-1;
a = map2(i,j);b = map2(i+1,j);c = map2(i+1,j+1); d = map2(i,j+1);
plotconn = [a(:) b(:) c(:) d(:)]; clear a b c d
plotconn = plotconn(~sum(plotconn==0,2),:); %for Plotting

%Create Chebyshev Matrix
D = -ChebydiffMatrix(N);
Dx = kron(eye(N+1),D); %2D x
Dy = kron(D,eye(N+1)); %2D y
```

```
I = repmat(permute(conn,[1 3 2]),1,(N+1)^2);
J = repmat(permute(conn,[3 1 2]),(N+1)^2,1);
Xx = repmat(2*Dx*E/lx,1,1,size(conn,2));
Xy = repmat(2*Dy*E/lx,1,1,size(conn,2));
Kx = sparse(I(:),J(:),Xx(:));
Ky = sparse(I(:),J(:),Xy(:));
clear I J Xx Xy
f = accumarray(conn(:),1);
fix = speye(length(f)).*f;
Dx = fix\Kx;
Dy = fix\Ky;
D2 = Dx^2+Dy^2;


exit = double(px');
entry = double(pe');
hwall = double(hw);
vwall = double(vw);
interior = setdiff(1:num_nodes,[exit entry hwall vwall]);

if nargin == 0
patch('Faces',plotconn,'Vertices',X','FaceColor','none','edgealpha',1);
set(gca,'XColor','none')
set(gca,'YColor','none')
caxis([-1e4 1e4]); axis equal
set(gca,'xtick',[])
set(gca,'ytick',[])
end
```

Main Method

```
E = 1;
N = 8;
lx = 1;
ly = 1;
rng(55);
% map = M(2:end-1,3:end-2); map = map==2;map=~map; map([1 end]) = 1;
map = [1 0;1 0;1 1 ; 0 1;1 1;1 0 ;1 0];
% map = rot90(double(~imread('map - Copy.png')),3); map = map(:,:,1);
[Dx,Dy,D2,D,n,X,conn,plotconn,exit,entry,hwall,vwall,interior] =
meshgen(E,N,lx,ly,map);
inu = @(x) 1-(2/3*(x-1.5)).^8; %0-3
inu = @(x) (1-(2*x-1).^8)*.1; %0-3
inu = @(x) (x-x.^2); %0-1
% inu = @(x) 1-(x/25-1).^50;
fprintf('Mesh Generated\n');
clf

x = (X(1,:))';
y = (X(2,:))';
therm = find(x>=-1 & x<=0 & y<=1 & y>=0);

fx = exp(-100*((x-.5).^2+(y-.5).^2))/100; fy = -0*fx;
exn = 1:length(exit);
u = zeros(n,1); v = zeros(n,1); p = v; h = v;
```

```
I = speye(n);Z = I*0;
dt = 0.001; t = 0;
nu = 1;
k_s = 1;

%POI
HW = sparse(hwall,hwall,1,n,n);
VW = sparse(vwall,vwall,1,n,n);
EX = sparse(exit ,exit ,1,n,n);
EN = sparse(entry,entry,1,n,n);
TB = sparse(therm,therm,1,n,n);
IN = sparse(interior,interior,1,n,n);
%EXN = sparse(exn,exit,1,exn(end),n)-sparse(exn,entry,1,exn(end),n);
U = speye(n).*u;
V = speye(n).*v;

ifin = @(x,y) logical(map(sub2ind(size(map),floor(x/lx)+1,floor(y/ly)+1)));

% vid = VideoWriter('Test5.mp4', 'MPEG-4');vid.Quality = 90;open(vid);
for k = 1:10

    %Inflow/Outflow
    u(entry) = inu(y(entry));
    U = speye(n).*u;
    V = speye(n).*v;
    fprintf('Iteration %i:\t',k);
    %Divergence Feild
    div = 1/dt*(Dx*u+Dy*v);%-(Dx*U*Dx*u+Dy*V*Dy*v);
    LHS = IN*D2 + VW*Dx + HW*Dy + EN*Dx + EX;
    rhs = IN*div;
    p = LHS\rhs;%[LHS;sparse(1,n,1,1,n)]\[rhs;0];
    fprintf('P Solved,\t');
    %Initiate U push
    rhs = (I/dt + nu*D2/2 - U*Dx/2 - V*Dy/2)*u - Dx*p;
    LHS = (I/dt - nu*D2/2 + U*Dx/2 + V*Dy/2);
    LHS = IN*LHS + VW + HW + EN + EX*Dx;
    rhs = IN*rhs+EN*inu(y);
    u = LHS\rhs;
    fprintf('U Solved,\t');
    %Initiate V push
    rhs = (I/dt + nu*D2/2 - U*Dx/2 - V*Dy/2)*v - Dy*p;
    LHS = (I/dt - nu*D2/2 + U*Dx/2 + V*Dy/2);
    LHS = IN*LHS + VW + HW + EN + EX*Dx;
    rhs = IN*rhs;
    v = LHS\rhs;
    fprintf('V Solved,\t');
    %Initiate H push
    rhs = (I/dt + k_s*D2/2 - U*Dx/2 - V*Dy/2)*h;
    LHS = (I/dt - k_s*D2/2 + U*Dx/2 + V*Dy/2);
    LHS = IN*LHS + VW*Dx + HW*Dy + EN + EX*Dx;
    LHS = LHS - TB*LHS + TB;
    rhs = IN*rhs + TB*ones(n,1);
    h = LHS\rhs;
    fprintf('T Solved,\t');

    %Initiate T push
```

```
    t = t + dt;
    vm = (u.^2+v.^2).^.5;
    clf

    %Plot feild
    subplot(2,1,1)

patch('Faces',plotconn,'Vertices',X','FaceVertexCData',h,'FaceColor','interp'
,'edgealpha',0.0);
    title(sprintf('Velocity Vector (Scaling Factor: 0.3x) and Temperature t =
%.3f',t)); caxis([0 0.25])
    hold on; set(get(colorbar,'label'),'string','Normalized Temperature');
    poi = find(sum(x==(0:13),2));
    sf = 0.3;
%
quiver(x(poi),y(poi),u(poi)*sf,v(poi)*sf,'w','showarrowhead','off','AutoScale
','off');
    hold off;  axis equal;    axis([min(x) max(x) min(y) max(y)])

    subplot(2,1,2);

patch('Faces',plotconn,'Vertices',X','FaceVertexCData',p,'FaceColor','interp'
,'edgealpha',0.0);
    title(sprintf('Pressure Correction Feild'));  xlabel('x-axis'); %caxis([-
1 3.5]);
    colorbar; axis equal;    axis([min(x) max(x) min(y) max(y)])
%     writeVideo(vid,getframe(gcf));
    drawnow;
    fprintf('Conserved: %.5f, MaxExit: %f\n',0, max(u(x==max(x))))
end
% close(vid);
```