Create polygon through fans suitable for parallel calculations

Kang Yang, Kevin Yang, Shuang-ren Ren Zhao
Imrecons Inc, Toronto Canada

Abstract

There are many methods for finding whether a point is inside a polygon or not. The congregation of all points inside a polygon can be referred to as the point congregation of the polygon. Assume on a plane there are N points. Assume the polygon has M vertices. There are $O(NM)$ calculations to create the point congregation of the polygon. Assume $N >> M$, we offer a parallel calculation method which is suitable for GPU programming. Our method considers a polygon is composed of many fan regions. The fan region can be positive and negative.
I. INTRODUCTION

There are methods to find out whether a point is inside a polygon or not. All points inside a polygon are the point congregation of polygon. If a plane have $N$ points and the polygon have $M$ vertices, If $N >> M$, found all points inside the polygon need $O(MN)$ calculations. If $N$ is very big, the above method to create the polygon is time consume. In case we have GPU, would like to find a method can parallel find all points inside the polygon.

By notice that a polygon can be built by positive/negative fan regions, we offers the following method. In this article we use Julia programming language to test our ideas.

II. HALF PLANE

Any two points can create a line, all points at the right of the line is a half plane. We use Julia programming language to test our idea. The following is to start the julia program,

```julia
using TestImages
using Images, Colors, FixedPointNumbers, ImageView
```

Assume there are 2 points $R_0$ and $R_1$, $R_0$, $R_1$ are vector. $R_0$ and $R_1$ can create a line. Any point in the plane is Point $P$. $P$ is vector. $P$ has two components $P[1] = i$, $P[2] = j$. Hence $P = [i, j]$. Assume $M$ is the direction vector from $R_0$ to $R_1$, i.e.,

$$M = R_1 - R_0$$

The normal vector on right side of the line is $N$,

$$N = [-M[2], M[1]]$$

Hence there is,

$$N = [-R_1[2] - R_0[2], R_1[1] - R_0[1]]$$

Define a vector $X$,

$$X = P - R_0$$
Figure 1: Half plane is created from 2 points $R_0$ and $R_1$.

We define if the value $v$

$$v = N \cdot X \geq 0$$

the point is $P$ is inside the half plane. Other points are not inside the half plane. Here “$\cdot$” inner product of two vector. We will use color green to show the point inside the half plane. We use color red to show the point outside the half plane. This half plane is at the right side of the line. We also give a value 1 to all green point. The other point give a value 0.

The following gives the Julia programming code for the half plane, see Figure (1).

The following is the function of half plane $H(R_0, R_1)$ which have 3 parameters. The first Point $R_0$, the second point $R_1$, and the image size $imsize$. The boundary line is includes inside the half plane.

```
function half_plane(R0,R1,imsize)
    n_vector=[-(R1[2]-R0[2]),R1[1]-R0[1]]
    B=zeros(imsize)
    for jji=1:imsize[2]
        for iii=1:imsize[1]
            x_vector=[iii-R0[1],jjj-R0[2]]
            value=n_vector'*x_vector
            if value[1,1] >= 0.
                B[iii,jjj]=1.0
            else
                B[iii,jjj]=0.0
            end
        end
    end
    return B
end
```
\[ B[iii, jjj] = 0. \]

end
end
end

\text{copy}(B)
end

If the boundary line does not include inside the half plane, the above formula need to be adjusted as following.

\[ v = N \cdot X > 0 \]

We call this is half plane less \( HL(R_0, R_1) \). The source code after this change becomes,

\begin{verbatim}
function half_plane_less(R0,R1,imsize)
    n_vector=[(R1[2]-R0[2]),R1[1]-R0[1]]
    B=zeros(imsize)
    for jjj=1:imsize[2]
        for iii=1:imsize[1]
            x_vector=[iii-R0[1],jjj-R0[2]]
            value=n_vector'*x_vector
            if value[1,1]>0.
                B[iii,jjj]=1.0
            else
                B[iii,jjj]=0.
            end
        end
    end
    copy(B)
end
\end{verbatim}

The following is the test program.
Figure 2: (a) Full plane. (b) Half plane created from 2 Points $R_0$ and $R_1$.

```plaintext
imsise=(600,500)
B0=ones(imsise)
my_view_flip(B0)
RR0=[100,100]
RR1=[400,100]
imsise=(600,500)
B1=half_plane(RR0,RR1,imsise)
my_view_flip(B1)
```

Figure(2) shows full plane and half plane which is created from 2 points R0 and R1.

III. FAN REGION

The two half can create a fan region. Assume we have 3 points. $R_0, R_1$ can define a half plane $H(R_0, R_1)$, $H(R_1, R_0)$ and $HL(R_0, R_1)$, $HL(R_1, R_0)$ as before. It is same to the points $R_1$, $R_2$. One half plane and one half plane less can create a fan region. See, Figure(3).

Details of definition is following

$$F(i,j) = HL(R_1, R_0) \cap H(R_1, R_2)(+1) \quad if \ v \geq 0$$

$$F(i,j) = HL(R_0, R_1) \cap H(R_2, R_1)(-1) \quad if \ v < 0$$

Where

$$v \equiv N \cdot X$$
Figure 3: (a) positive fan with Green color, (b) negative fan with red color.

\[
X = X_2 - X_1
\]

\[
N = [-R_1[2] - R_0[2], R_1[1] - R_0[1]]
\]

\( F \) is all points inside the fan region. The above formula means if \( R_2 \) inside the half plane \( H(R_0, R_1) \) the fan’s value is negative. If \( R_2 \) is not inside the half plane \( H(R_0, R_1) \) the fan’s value is negative. The Julia source code is following,

```julia
# calculate the fan image:
function fanregion(R0,R1,R2,imsize)
    n_vector=[-(R1[2]-R0[2]),R1[1]-R0[1]]
    x_vector=R2-R1
    value=n_vector'*x_vector
    if value[1,1]>=0.
        B=half_plane(R0,R1,imsize).*half_plane_less(R2,R1,imsize)
        B*=-1.0
    else
        B=half_plane_less(R1,R0,imsize).*half_plane(R1,R2,imsize)
        B*=-+1.0
    end
    copy(B)
end
```
Figure 4: (a) A negative fan region created from $R_0, R_1$ and $R_2$. (b) A positive fan region created from points $R_1, R_2$ and $R_3$.

The following is the program with 4 points to test the fan region. We create two fan regions.

```
RR0=[100,100]
RR1=[400,100]
RR2=[400,200]
RR3=[500,300]
RR4=[200,500]
imsiz=(600,500)
B2=fanregion(RR0,RR1,RR2,imsiz)
my_view_fi(B2)
B3=fanregion(RR1,RR2,RR3,imsiz)
my_view_fi(B3)
```

IV. POLYGON

Polygon can be built from all fan region, some is positive some is negative. Assume we have 5 points $R_0, R_1, R_2, R_3, R_4$

$Polygon = F(R_0, R_1, R_2) + F(R_1, R_2, R_3) + F(R_2, R_3, R_4) + F(R_3, R_4, R_0) + F(R_4, R_0, R_1) + 1$
Figure 5: (a) The polygon created from 5 points.

The Julia program code is following,

```
polygon=ones(imsize)+
    fanregion(RR0,RR1,RR2,imsize)+
    fanregion(RR1,RR2,RR3,imsize)+
    fanregion(RR2,RR3,RR4,imsize)+
    fanregion(RR3,RR4,RR0,imsize)+
    fanregion(RR4,RR0,RR1,imsize)
my_view_flip(polygon)
```

The general function to created a polygon is given as following.

```
function my_polygon(points,image_size)
    imsize=size(points)
    R0=points[:,end]
    R1=points[:,1]
    R2=points[:,2]
    poli=fanregion(R0,R1,R2,image_size)
    my_view(poli)
    for iii=1:imsize[2]-1
        R0=points[:,iii]
```
iii_1=iii+1
R1=points[:,iii_1]
iii_2=iii+2
if iii_2>imsize[2]
    iii_2=imsize[2]
end
R2[1]=points[1,iii_2]
R2[2]=points[2,iii_2]
fan=fanregion(R0,R1,R2,image_size)
poli+=fan
my_view(poli)
end
poli+=1.
my_view(poli)
return poli
end

V. IMAGE VIEWER

In order to view the image, we have the following functions. We have flipped the image along for y coordinates, so y coordinates directed above.

function flip_y(B)
    imsize=size(B)
    D=copy(B)
    size_y=imsize[2]
    half_size=Int(floor(size_y/2))
    for jjj=1:half_size
        for iii=1:imsize[1]
            D[iii,size_y-jjj]=B[iii,jjj]
            D[iii,jjj]=B[iii,size_y-jjj]
        end
    end
end
function my_view_ip(B)
    #only the size of B is
    D=[RGBU8(0,0,0) for iii=1:size(B)[1], jjj=1:size(B)[2]]
    for jjj=1:size(B)[2]
        for iii=1:size(B)[1]
            if B[iii,jjj]>0.5
                D[iii,jjj]=RGBU8(0,1,0)
            elseif B[iii,jjj]<-0.5
                D[iii,jjj]=RGBU8(1,0,0)
            else
                D[iii,jjj]=RGBU8(0,0,0)
            end
        end
    end
    D=flip_y(D)  imgc = copyproperties(img, D)  view(imgc) end
function my_view(B)
    D=[RGBU8(B[iii,jjj],B[iii,jjj],B[iii,jjj]) for iii=1:size(B)[1], jjj=1:size(B)[2]]
    imgc = copyproperties(img, D)
    view(imgc)
    imgc
end

VI. TEST IMAGE

RR0=[100,100]
RR1=[400,100]
When the program my_polygon(points,imsize) runs, it shows how a polygon is created from the fan regions, see Figure(6).

VII. CONCLUSION

Introduced a method to draw all points inside the polygon. The polygon divided as may fan region. Each time a fan is drawn. A fan is created by two half planes. The half plane is created through 2 points. This method is suitable parallel calculation for example GPU Cuda/OpenCL calculations.

Figure 6: (a) The polygon created from 5 points.


[11] Accurate point in triangle test "...the most famous methods to solve it"


