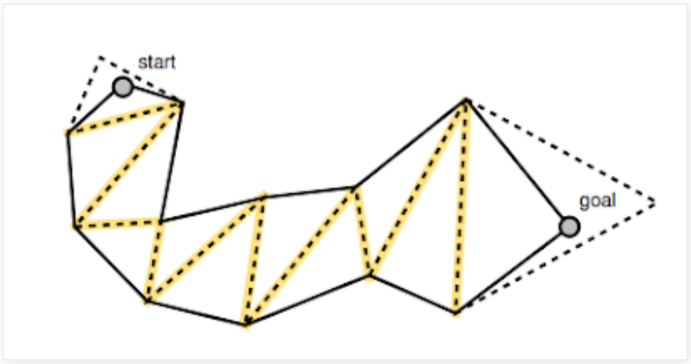
Digesting Duck

Blog about game AI and prototyping

Monday, March 8, 2010

Simple Stupid Funnel Algorithm

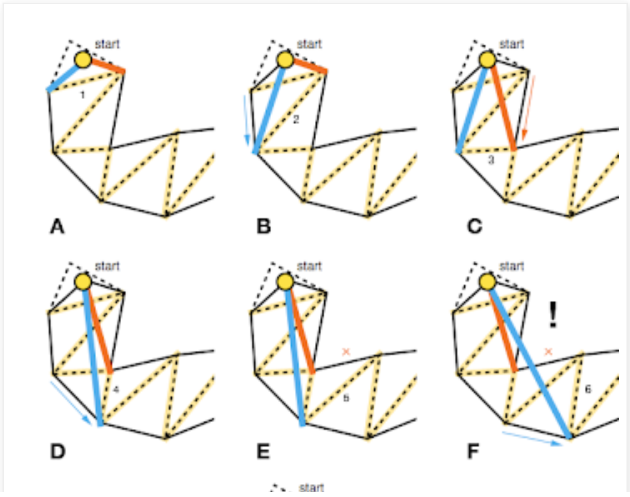


[EDIT] Fixed that example scenario below.

There's a nice master class over at AIGameDev.com about pathfinding: Hierarchical Pathfinding Tips & Tricks with Alexander Kring. That slide about Left 4 Dead path smoothing hit my pet peeve. It makes me sad that people still use "let's trace the polygon edge midpoints" as the reference algorithm for path smoothing. Really sad.

Funnel algorithm is simple algorithm to find straight path along portals. Of the most detailed descriptions can be found in Efficient Triangulation-Based Pathfinding.

Here's implementation of simple stupid funnel algorithm. Instead of using queue and all that fancy stuff, simple stupid funnel algorithm runs a loop and restarts the loop from earlier location when new corner is added. This means that some portals are calculated a bit more often than potentially necessary, but it makes the implementation a whole lot simpler.



Search This Blog

Search

Recast & Detour

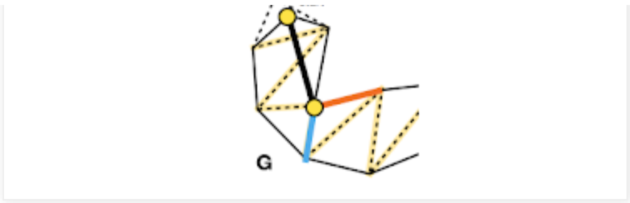
- Recast & Detour discussion group
- Recast & Detour source code

Blog Archive

- ► 2012 (7)
- ► 2011 (30)
- ▼ 2010 (70)
  - ► December (3)
  - ► November (4)
  - ► October (4)
  - ► September (2)
  - ► August (18)
  - ► July (8)
  - ► June (2)
  - ► May (4)
  - ► April (4)
  - ▼ March (10)
    - My Favorite Protyping Environment
    - Detour Serialization API
    - Geometric vs. Sampling
    - Custom hRVO
    - Local Navigation Grids
    - Simple Stupid Funnel Algorithm
    - Blind Sighted
    - Save Games and All That
    - Power of Less
    - Sketch for Hierarchical Pathfinding Using Detour
  - ► February (5)
  - ► January (6)
- ► 2009 (56)

Interesting Blogs

Diary of a Graphics Programmer



The above image shows six steps of the algorithm. Every time we process a new portal edge (the dashed lines highlighted in yellow), we first:

- Check if the left and right points are inside the current funnel (described by the blue and red lines), if they are, we simple narrow the funnel (A-D).
- If the new left endpoint is outside the funnel, the funnel is not update (E-F)
- If the new left end point is over right funnel edge (F), we add the right funnel as a corner in the path and place the apex of the funnel at the right funnel point location and restart the algorithm from there (G).

The same logic goes for left edge too. This is repeated until all the portal edges are processed.

As seen in the above example, the algorithm recalculates some edge several times because of the restart, but since the calculations are so simple, this simple stupid trick makes the implementation much simpler and in practice is faster too.

```
inline float triarea2(const float* a, const float* b, const float* c)
{
 const float ax = b[0] - a[0];
 const float ay = b[1] - a[1];
 const float bx = c[0] - a[0];
 const float by = c[1] - a[1];
 return bx*ay - ax*by;
}

inline bool vequal(const float* a, const float* b)
{
 static const float eq = 0.001f*0.001f;
 return vdistsqr(a, b) < eq;
}

int stringPull(const float* portals, int nportals,
         float* pts, const int maxPts)
{
     // Find straight path.
 int npts = 0;
     // Init scan state
 float portalApex[2], portalLeft[2], portalRight[2];
 int apexIndex = 0, leftIndex = 0, rightIndex = 0;
 vcpy(portalApex, &portals[0]);
 vcpy(portalLeft, &portals[0]);
 vcpy(portalRight, &portals[2]);

     // Add start point.
 vcpy(&pts[npts*2], portalApex);
 npts++;

 for (int i = 1; i < nportals && npts < maxPts; ++i)
 {
     const float* left = &portals[i*4+0];
     const float* right = &portals[i*4+2];
```

Coder Corner

Stumbling Toward 'Awesomeness'

Gaffer on Games

Game AI for Developers

Atom

realtimecollisiondetection.net - the blog

Game/AI

Chris Hecker's Website

About Me

Mikko Mononen

Finland

View my complete profile

```
        // Update right vertex.
       if (triarea2(portalApex, portalRight, right) <= 0.0f)
 {
        if (vequal(portalApex, portalRight) || triarea2(portalApex, portalLeft, right) > 0.0f)
        {
             // Tighten the funnel.
            vcpy(portalRight, right);
            rightIndex = i;
        }
        else
        {
                // Right over left, insert left to path and restart scan from portal left point.
            vcpy(&pts[npts*2], portalLeft);
            npts++;
                // Make current left the new apex.
            vcpy(portalApex, portalLeft);
            apexIndex = leftIndex;
                // Reset portal
            vcpy(portalLeft, portalApex);
            vcpy(portalRight, portalApex);
            leftIndex = apexIndex;
            rightIndex = apexIndex;
                // Restart scan
            i = apexIndex;
            continue;
        }
    }

        // Update left vertex.
       if (triarea2(portalApex, portalLeft, left) >= 0.0f)
 {
        if (vequal(portalApex, portalLeft) || triarea2(portalApex, portalRight, left) < 0.0f)
        {
                // Tighten the funnel.
            vcpy(portalLeft, left);
            leftIndex = i;
        }
        else
        {
             // Left over right, insert right to path and restart scan from portal right point.
            vcpy(&pts[npts*2], portalRight);
            npts++;
                // Make current right the new apex.
            vcpy(portalApex, portalRight);
            apexIndex = rightIndex;
                // Reset portal
            vcpy(portalLeft, portalApex);
            vcpy(portalRight, portalApex);
            leftIndex = apexIndex;
            rightIndex = apexIndex;
                // Restart scan
            i = apexIndex;
            continue;
        }
    }
}
    // Append last point to path.
if (npts < maxPts)
{
    vcpy(&pts[npts*2], &portals[(nportals-1)*4+0]);
    npts++;
 }

 return npts;
}
```

The array *portals* has all the portal segments of the path to simplify, first left edge point and the right edge point. The first portal's left and right location are set to start location, and the last portals left and right location is set to end location of the path. Something like this:

```
// Start portal
vcpy(&portals[nportals*4+0], startPos);
vcpy(&portals[nportals*4+2], startPos);
nportals++;
// Portal between navmesh polygons
for (int i = 0; i < path->npolys-1; ++i)
{
 getPortalPoints(mesh, path->poly[i], path->poly[i+1], &portals[nportals*4+0], &portals[nportals*4+2]);
 nportals++;
}
// End portal
vcpy(&portals[nportals*4+0], endPos);
vcpy(&portals[nportals*4+2], endPos);
nportals++;
```

The cool thing is that this algorithm can be used with pretty much any navigation format. Let it be a grid, quad-tree, connected squares, way-points or navigation mesh. As long as you pass it a list of line segments that describe the portal from one node to another.

This algorithm is also awesome when combined with steering. You can calculate the desired movement velocity by calculating the next few turns and the steer towards the next corner. It is so fast that you can calculate this every iteration just before you use your favorite steering code.

So there you have it. The next person who goes public and suggest to use edge midpoints and raycasting to find more straight path after A* will get his inbox flooded with Cute Overload weekly selection to instigate his pet hate too.

Posted by Mikko Mononen at 2:45 AM

69 comments:

Anonymous March 8, 2010 at 6:47 AM

I can think of one case where funneling does not quite work, which ultimately made me write the very technique of raycasting midpoints for use within Recast/Detour on my modified version, and it's a nice coincidence you're writing about this as I hit this very problem late last week.

There can be a nasty horizon effect happening if the polygons found by A* do not themselves form a straight/smooth path to the goal. This seems to get even worse when you arbitrarily tesselate the navmesh to create areas (I wrote my own implementation a few months ago not knowing about your solution, I have not looked yet by how much they differ), where the polygons you select might have as boundary some of these other extraneous polygons that are part of the straight path, but have not been retained by A* because their midpoints were too far (most often because of a big polygon). So you end up with a path that has these strange sharp turns in the middle of a relatively open field. Raycasting has at least the advantage of finding a straighter path regardless of the A* result, not quite perfect because sometimes the next midpoint temporarily deviates behind an obstacle while the next one is visible, but this approach still solved most of the problems I had before.

I did think of reusing the polygons found by raycasting to the funnel algorithm, but another side effect of smoothing midpoints that I found interesting is that it often makes the path prefer to stay a little further away from obstacles (provided the triangles are not too large, but this case could be handled by sampling), rather than tightly hugging them for a sharp turn, which looks a bit more natural and leave some breathing room. Of course this is a problem possibly handled better in the path follow logic, but given the time constraints, for now it works well enough. Maybe however using the modified funnel algorithm from this paper could be interesting, though the radius would not be a strict exclusion, more like a preferred stay-away-from-obstacles distance, but I have not looked yet if this desired behavior is compatible with this algorithm or your simplified implementation.

Of course I might have overlooked something, but this was the best solution I could think of within a quite short time frame. I'm curious to see if you've encountered this problem before and what are your thoughts about it.

Reply

Mikko Mononen March 8, 2010 at 7:13 AM

That is not problem with funnel algorithm, but A*. Ideally you should not fix the path smoothing (because it works), but actually add extra step after A* which tries to adjust the list of path polygons so that it will create shorter path.

You should spend absolutely minimum amount of time trying to create smooth path, because it will change when someone else moves in the game world.

Stuff like boundary hugging are better handled using steering algorithms or such, you just get better results that way.

Just to reiterate:

1) A* transforms graph into a linear set of polygons which will leave you to target.
2) String pulling will tell you where to towards in order to move towards target
3) Locomotion/steering will be in charge of the final smooth path.

That way you will do as little work as necessary. Every smarter-than-potatoe string pulling out there which tries to solve the whole path following dilemma at that stage, just does too much work, and usually creates not very good results, which can described as "more natural and human like" ;)

Reply

Unknown March 8, 2010 at 12:14 PM

Great post! Could you talk a bit more about how you account for the agent radius size? The way I handled this was to iterate over the final list of smoothed points, and move them further away from the edges of the nav mesh. The question is, in what direction do you move the points, to get them away from the edge of the nav mesh? Here is a bit of psuedocode to describe the direction I use for the offset:

```
a = points[i+1] - points[i]
b = points[i+2] - points[i]
c = Normalize(a*05 + b*0.5)
if (ClockwiseTo(b,a))
{
offsetDir = Cross(c,up)
}
else
{
offsetDir = Cross(up,c)
}
points[i+1] += offsetDir*agentRadius
```

However, I can see some edge cases where this would not give the best position to move toward. How did you handle this problem through steering?

Reply

Mikko Mononen March 8, 2010 at 1:02 PM

This implementation assumes that the portals are already adjusted so that they contain the agent radius. For example, a navmesh would be shrinked by agent radius.

This assumption reduces the complexity of all the thing you need to do with navmeshes or any navigation representation a huge amount.

The only corner case for steering is when you are really close to a corner. AFAIK there is no good solid way to handle that. I usually build the system so that I allow the agent to move quite close to the corner (say, 0.1cm) and then choose the next corner instead. The movement code then deals with this extra slop. You can check the Recast Demo for an example how to do this.

Reply

Jad Nohra March 15, 2010 at 3:16 AM

by 'let's trace the polygon edge midpoints' do you mean "Reactive Path Following" in the l4d powerpoint?

Reply

Mikko Mononen March 15, 2010 at 3:29 AM

Yes. That 'let's trace the polygon edge midpoints' seems to be a common plague.

I have used string pulling as coder interview questions for quite some time (thanks Danny!) and 60% of the results have been 'let's trace the polygon edge midpoints' plus some wonky magic.

Only one person (who actually had no AI background) so far has submitted a version of funnel algorithm.

Reply

Mikko Mononen March 15, 2010 at 3:49 AM

My point was also, that developers should know better, and not compare their new method to 'let's trace the polygon edge midpoints'.

It is like someone would compare Killzone2 graphics to a flat shaded triangle.

Reply

Anonymous March 18, 2010 at 7:38 PM

how do u do?

Reply

Cameron Hart April 1, 2010 at 8:51 PM

Mikko, in one of your comments you say, "Ideally you should not fix the path smoothing (because it works), but actually add extra step after A* which tries to adjust the list of path polygons so that it will create shorter path". I'm curious as to how that would work? The only method that comes to my mind is effectively string pulling to collapse redundant path nodes.

Reply

Mikko Mononen April 2, 2010 at 12:30 AM

Cam, I have not put too much thought into that. The first thing to check would be to see if the extra detour always happens with same pattern. For example, the problems with A* and polygons pretty much always happens when you have large and small next to each other.

You could use the straight path found by the funnel algorithm as "heuristic" to see how well some alternative corridor performs. The TRA* paper has some discussion regarding this.

My point was that you should keep the string pulling as fast as possible since you will be updating it constantly when you follow the path.

If the path corridor is not good enough, then you should fix the problem at its root, not higher up where it takes more processing to get the thing done.

Reply

Cameron Hart April 2, 2010 at 8:24 PM

Yeah you are probably right about the path corridor being wrong. It is generally happening in an open area with lots of tiles. What I'm seeing is basically similar to the bottom image in this image http://3.bp.blogspot.com/_-u6ZJlBFOL0/Sl8NQfhU_pI/AAAAAAAAADI/DS3h8mkIljc/s1600-h/heuristics_take2.jpg

Maybe not the end of the would but if the agent is quite small relative to the tile it's kind of obvious that they're taking a slightly askew route. So it's this sort of scenario I'm interested in solving. On a related note, the post you wrote on Recast's generation settings was quite helpful, but it didn't mention tile sizes. Have you written anything about choosing a good tile size? Mostly I've been using tiles to limit how long edges can get, not for dynamic meshes. But maybe this is the wrong approach.

Reply

Mikko Mononen April 3, 2010 at 9:15 AM

The tile size depends on why you use tiles in the first place. I shall write a post about it.

That bottom image is actually pretty good, consider yourself lucky ;)

The failure pattern in your case that string pulled path* contains "inner mesh" vertex. If you encounter that, you could use raycast() to check if there is straight line to the straight path vertex just after the inner vertex.

If there is, then you replace the beginning of your path corridor by the results returned by raycast(). Basically you find the furthest common polygon and batch the path corridor up to that polygon.

You should also have a cache of false alarms, so that you don't try to short cut if the test failed last time.

There might be other ways to handle that, like trying to find another path around the bad vertex, but I'm quite positive that you can use that inner vertex check to see when you should try to adjust the path. I'm sure there will be other cases too.

*) I usually query just next 2-3 straight path vertices when following path, that idea should work with that kind of queries stuff too, you kinda fix the path as you go.

Reply

Mikko Mononen April 3, 2010 at 9:42 AM

One more thing: note that the raycast() is used to fix the corridor, _not_ to fix the straight path.

Reply

Cameron Hart April 3, 2010 at 1:42 PM

Ah cool. That's what I've done in the past. I was under the impression that was what you were objecting to, but I think I got the wrong end of the stick :)

Thanks for your help.

Reply

Mikko Mononen April 3, 2010 at 2:00 PM

Just make sure you fix the corridor using the raycast, and not use that to fix string pulling every time you need new straight path. Then I don't object :)

Reply

Mene June 25, 2010 at 7:17 AM

This comment has been removed by the author.

Reply

Mene June 25, 2010 at 7:19 AM

This comment has been removed by the author.

Reply

Mene June 25, 2010 at 7:25 AM

Don't you need to check if the destination point lies above or below the current funnel? If so you need to add the left / right point as additional waypoint. Or am I missing something?
Consider this case: http://bit.ly/aY0YHV
(purple is current apex, orange arrow is left, blue arrow is right, yellow arrows are portals, target is the circle, beginning of the channel is top left)

kiitos for this blog ;D

Reply

Mikko Mononen June 25, 2010 at 8:59 AM

In your example, the point will be added, when you process the last portal, that is, the destination point. If you take a look at the usage code above, I set the last portal to be null length segment, where both end-points are the goal point.

Reply

Mene June 25, 2010 at 11:06 AM

Ah, so I indeed did miss something.

Reply

Unknown December 6, 2010 at 3:23 PM

Hi,

Can you please tell me what does vcpy do?
Does it copy vectors?

Thanks in advance

Reply

Mikko Mononen December 7, 2010 at 12:00 AM

For 3d vector:

```
inline void vcpy(float* dst, const float* src)
{
dst[0] = src[0];
dst[1] = src[1];
dst[2] = src[2];
}
```

for 2D vector

```
inline void vcpy(float* dst, const float* src)
{
dst[0] = src[0];
dst[1] = src[1];
}
```

Reply

Unknown December 30, 2010 at 11:00 AM

I would really like to understand what triarea2 does, but my geometry/trig/linear algebra skills are very rusty. Can you describe this formula enough so that I can search around and figure out how this works?

Reply

Mikko Mononen December 30, 2010 at 11:22 AM

It calculates twice the triangle area using 2D cross product. See the following article for more details: http://softsurfer.com/Archive/algorithm_0101/algorithm_0101.htm

Reply

soywiz February 17, 2011 at 3:27 AM

Javascript implementation of "Digesting Duck :: Simple Stupid Funnel Algorithm":

http://pastebin.com/7jwrmw1i

Reply

utkutest April 19, 2011 at 3:48 PM

Hi Mikko,

Portals do not have to be between triangles, portals between convex polygons with any number of vertices will also work right ?

Reply

Mikko Mononen April 19, 2011 at 11:10 PM

Hi utkutest, portals can be between any convex shape. The only restriction is that the portals should not intersect, by they can share a common vertex, though. This means that the string pulling can be also used for grid based pathfinders. Squares are as convex polygons as others :)

Reply

utkutest April 21, 2011 at 8:45 AM

Thanks Mikko, I've read the paper, your implementation is way cooler to be honest, I did not like that pseudo code at all, it was even very hard to follow for me.

Reply

Stephen October 10, 2011 at 9:29 AM

Hi,

Thanks for this post, I was trying to solve this problem and you're approach is working well.

I'm curious about the agent radius problem, you say in an earlier post that:

"This implementation assumes that the portals are already adjusted so that they contain the agent radius. For example, a navmesh would be shrinked by agent radius."

But what does it mean in practice to shrink to mesh?

I notice that thesis you linked to has it's own suggestions for dealing with agent size without modifying the mesh itself, do you recommend this approach?

Thanks for any help.

Reply

c October 11, 2011 at 3:34 PM

I was wondering the same thing as Stephen.
And if you preshrink a mesh does this mean that all agents have to have the same radius?

Reply

Mikko Mononen October 11, 2011 at 10:30 PM

c, yes. The whole problem is about 100x more complicated if you don't shrink the mesh. There are links to such papers elsewhere on this blog.

Reply

c October 12, 2011 at 1:03 AM

The disadvantage is that shrinking the mesh isn't so easy if you have to build the mesh by hand, especially if you have to build multiple meshes for different agent sizes.

Reply

Mikko Mononen October 12, 2011 at 5:21 AM

c, that's true. I recommend automatic navmesh building :)

Reply

c October 12, 2011 at 11:11 AM

Even though I'm working in 2D and not 3D, I think automatic navmesh generation is a little beyond my abilities, especially since I cannot find anything on the subject on the internet.

Reply

Mikko Mononen October 12, 2011 at 11:15 AM

c, this is your lucky day! Look around this blog, it's all about automatic navmesh generation.

Reply

c October 12, 2011 at 12:21 PM

Maybe, but I'm having a hard time finding anything and most of the stuff you write on this site is over my head.

Reply

Unknown November 27, 2011 at 12:54 PM

When checking the left portion do we have to compare angles counterclockwise? And when it's the right portion is it clockwise?

Reply

Mikko Mononen November 27, 2011 at 11:28 PM

@Unknown, you don't need to compare angles. You just need to check if a point is on a certain side of a segment. This can be done by calculating the winding of a triangle which is formet by the two points of the segment and the third point you want to check. The sign of the triangle area will tell you on which way the triangle is would, that is, on which side of the segment the third point is.

Reply

Unknown December 8, 2011 at 7:05 PM

Thanks a lot. What happen if triarea = 0 ?? The point tested goes through both if conditions?

Reply

Unknown January 2, 2012 at 6:35 AM

This comment has been removed by the author.

Reply

Unknown January 2, 2012 at 6:35 AM

This comment has been removed by the author.

Reply

Unknown January 2, 2012 at 6:36 AM

Does anyone know the problem in this case?
http://www.gamedev.net/topic/617458-doug-demyens-funnel-algorithm/

Reply

shawn May 23, 2012 at 10:56 AM

hi Mikko,

some game developers pointed out that n-sided convex polygon mesh is not ideal for pathfinding. They thought tri-mesh is better. but i thought the search can be faster in a poly-based mesh as less nodes are examined when you treat them as A* nodes. what do you think? do you think the poly-based mesh is still worth studying?

I just implemented your methods on a polygon-based mesh. Compared with triangle-based mesh, I found the running time even slower. So I am confused now.

cheers,
Shawn

Reply

Mikko Mononen May 23, 2012 at 11:03 AM

Polygons are indeed usually favored since they result less A* nodes. That is the reason Detour uses them too. The memory usage is usually similar.

What are those dev's reasons for tri meshes being better?

What parts of code are slower when you use polygon based methods?

Reply

Replies

Xiao May 30, 2012 at 6:34 AM

Thanks a lot, Mikko. Your comment is quite helpful.

I thought the problem could be how to find the left and right points. In a triangle-based mesh, if we know a point is on one side, the other point is clearly on the opposite side. Cycling through the triangles is much faster than the polygons as we have to decide the left and right points by cycling through all the end-points in the polygon.

For example, to determine whether a point on a portal, we have to cycle through the polygon from that point in two directions until finding a previous left or right point or another endpoint of that portal.

I thought this process takes more time than it in triangles. Do you have any suggestion to improve it?

And do you have any idea why many game engines still choose a tri-mesh based pathfinding? Does it imply that in the context of geometry, generating a triangle-based mesh (splitting a map into a set of triangles) is much easier or faster than generating a polygon-based mesh?

Many thanks again.

Cheers,
Shawn

Mikko Mononen May 30, 2012 at 6:39 AM

Finding portal between two polygons should not be a speed issue. Each edge stores an index/reference to neighbour polygon, so it is just few integer compares.

One advantage triangles have is that they are really easy to store, as each triangle takes the same amount of memory. Detour stores polygons in similar manner too to keep random access fast (i.e. all polygons have 6 vertices), but in the process you loose some memory. The amount of memory is usually insignificant.

Xiao May 30, 2012 at 10:22 PM

Yes, you are right. Finding a portal edge between two adjacent polygons is not a big issue. But we have to determine which point is left and right before we can process the portal.

I thought my program spent too much time on determining the left and right points (because each time I have to cycle through the whole polygon so as to determine which is left or right point based on previous left and right points). Do you have better idea?

Many thanks,
Shawn

Mikko Mononen May 30, 2012 at 10:30 PM

You're thinking too complicated. If an edge in a CCW triangle/polygon is defined as two consecutive vertices, then the first of those vertices is right and second vertex left. No need to calculate anything at all! :)

Xiao June 20, 2012 at 11:24 AM

Thanks a lot, Mikko.

As you have suggested, I stored polygons in CCW and did some pre-processing including:

- calculating adjacent polygons for each polygon
- finding the mutual edge between each two polygons

Now the program is much faster than triangle-based one. Thank you.

Cheers,
Shawn

Reply

Ollik June 28, 2012 at 7:29 AM

I did a test using pretty much a copy of your code. For me, the end point gets added twice to the result path in some cases.

More specifically this happens if the left or right vertex of the last actual portal is not part of the result path.

Not 100% sure if it's bug in the code or maybe something to do with the input data, but with a quick investigation my input data looked to be sane.

Reply

Anonymous October 16, 2012 at 8:38 AM

Hi Mikko, can you suggest an algorithm or reference for constructing the portals (or channels as some white papers call them)? I think I'm constructing them mostly correctly but missing a few edge cases.

Reply

Mikko Mononen October 16, 2012 at 9:08 AM

@Cameron, in my code and the pictures above, I call the yellow dashed edges portals. If you're polygons are wound counter clockwise, then it should be as simple as using the two consecutive vertices if the current polygon as the portal.

For example:
if pidx=current polygon, eidx=corrent polygon edges index (that is the edge that leads o the next polygon), then the portal is:

```
p = polygons[pidx];
left = vertices[p.verts[eidx];
right = vertices[p.verts[(eidx+1)%p.nverts];
```

(I may have left and right mixed up in the code, or the code may actually assume clockwise polygon, I tend to mix these up sometimes).

Reply

Replies

Anonymous October 16, 2012 at 10:37 AM

Thank you for the swift reply Mikko. That makes sense, I think I was over complicating it. Thanks again for the great write up!

Reply

Sean December 8, 2012 at 8:28 AM

So, I've been working on my own implementation of the funnel algorithm and I realized what sacrifice you're making with this (as you called it) "simple stupid funnel algorithm". The "sleeve" you've shown in the illustration is clearly taken from the Lee and Preparata paper. However, it works because the funnel's extents get contracted by each successive portal/edge. Your funnel doesn't cut through the center of portals. If it were to, you'd get an unhappy path. It is for these cases that the whole double-ended queue is used. Did you want to comment on that at all?

Reply

nixt December 23, 2013 at 4:46 PM

Which one is the left and right edges? Blue or Orange?

Its really confusing because in the given demo, you first say

If the new left endpoint is outside the funnel, the funnel is not update (E-F)
(indicating that orange is left)

and then you say:

If the new left end point is over right funnel edge (F), we add the right funnel as a corner in the path and place the apex of the funnel at the right funnel point location and restart the algorithm from there (G).
(indicating blue is left)

I've been trying to link the demo with the code for hours but it isn't making sense...

Reply

Mikko Mononen December 25, 2013 at 9:23 AM

The algorithm is quite confusing to try to explain in words. I think in the point 3/3 I have mixed left and right, so it probably should read:

If the new right end point is over left funnel edge (F), we add the left funnel as a corner in the path and place the apex of the funnel at the left funnel point location and restart the algorithm from there (G).

Reply

Unknown September 1, 2014 at 11:25 PM

thanx for this detail
Plastic Manufacturers

Reply

pslvseo a1 March 12, 2019 at 11:52 PM

A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive programming/company interview Questions.
website: geeksforgeeks.org

Reply

byodbuzz06 March 29, 2019 at 1:23 AM

A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive
programming/company interview Questions.
website: geeksforgeeks.org

Reply

byodbuzz06 March 31, 2019 at 9:00 PM

A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive
programming/company interview Questions.
website: geeksforgeeks.org

Reply

byodbuzz05 April 1, 2019 at 11:51 PM

A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive
programming/company interview Questions.
website: geeksforgeeks.org

Reply

byodbuzz05 April 3, 2019 at 9:13 PM

A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive
programming/company interview Questions.
website: geeksforgeeks.org

Reply

Digital Marketing August 1, 2019 at 2:47 AM

Excellent information, I like your blog.

Top 10 Business & Finance News in UK

Latest Business News in UK

Breaking News in UK

Reply

Anmol Rathi March 18, 2020 at 4:50 AM

This comment has been removed by the author.

Reply

Evan Luthra March 18, 2020 at 6:52 AM

Thanks for sharing:
Evan Luthra a young billionaire who was ahead of his time and is now investing in brilliant minds like him.Evan Luthra is one of the youngest tycoons in the IT industry today and has invested in numerous tech projects to this date. Evan Luthra started giving TedX speeches at the age of 18 and since then, he has also spoken at some of the most prestigious events including Crpto Investor Show, Blockshow, etc. He has been featured in 30 under 30 entrepreneurs list and is known to support and guide budding entrepreneurs through his incubator Startup Studio.

Having accumulated years of experience in the tech industry, Evan Luthra is today spearheading innovation in projects that are laying the foundation of our better future. He is not just a co-founder and COO of EL Group International, but he is also an active contributor in the global entrepreneurship community. Through Startup Studio, he is able to identify emerging businesses and share his business acumen that can drive these projects to success.

Although Evan Luthra invests in all kinds of tech projects, he focuses on blockchain projects as that sphere has become more of his specialization. He was instrumental in the success of blockchain projects like Relictum Pro, Crescent, and Relictum Pro.Evan Luthrais also influencing the world through his social media handles. With over 500,000 followers he is always sharing bits of his life and words of wisdom to let the world know that Evan Luthra is there to listen to your extraordinary ideas.

Reply

fovasly August 18, 2020 at 11:30 AM

Attention: Bloggers And Influencers
"Discover How You Can Use A Simple 'Funnel' To Add A Whole New Income Stream To Your Business...
https://bit.ly/FChallange

Reply

Meenakshi Khatri November 1, 2020 at 1:00 AM

Hey...Valuable information...Really helpful...Get Money Tips Here...

Reply

Unknown October 15, 2021 at 2:08 PM

Hello

Reply

jonathan ingram April 22, 2022 at 3:15 AM

This algorithm does not work in all cases. I have carefully implemented in full only to find it cannot work.

Please take this down.

Reply

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)