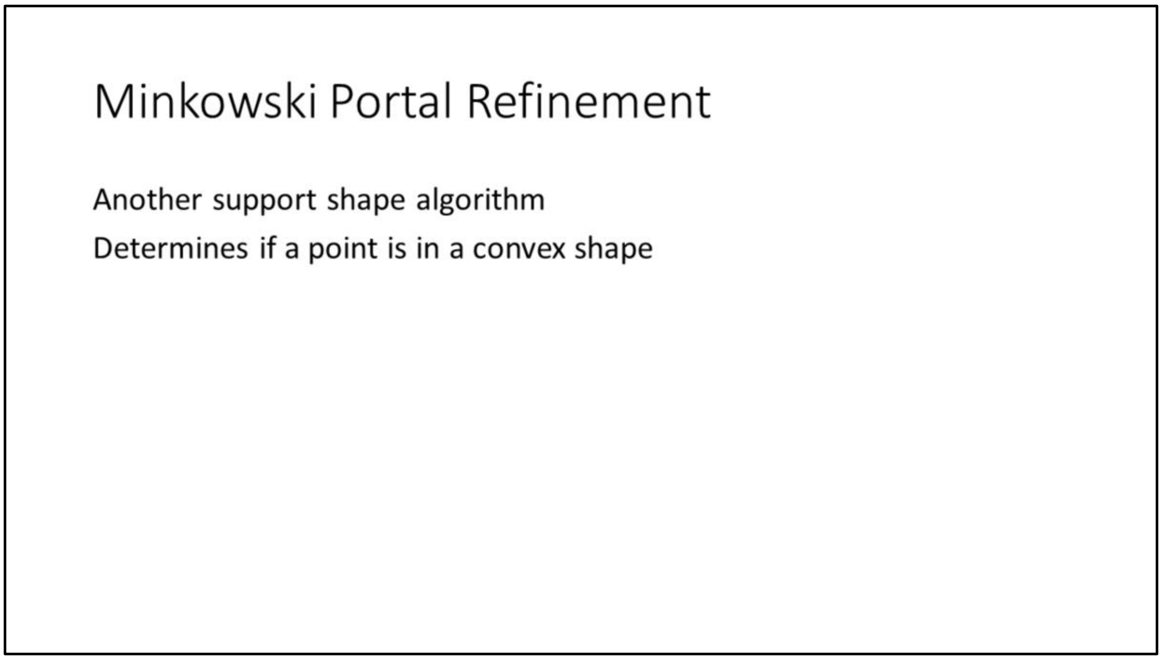
MPR

jodavis42@gmail.com

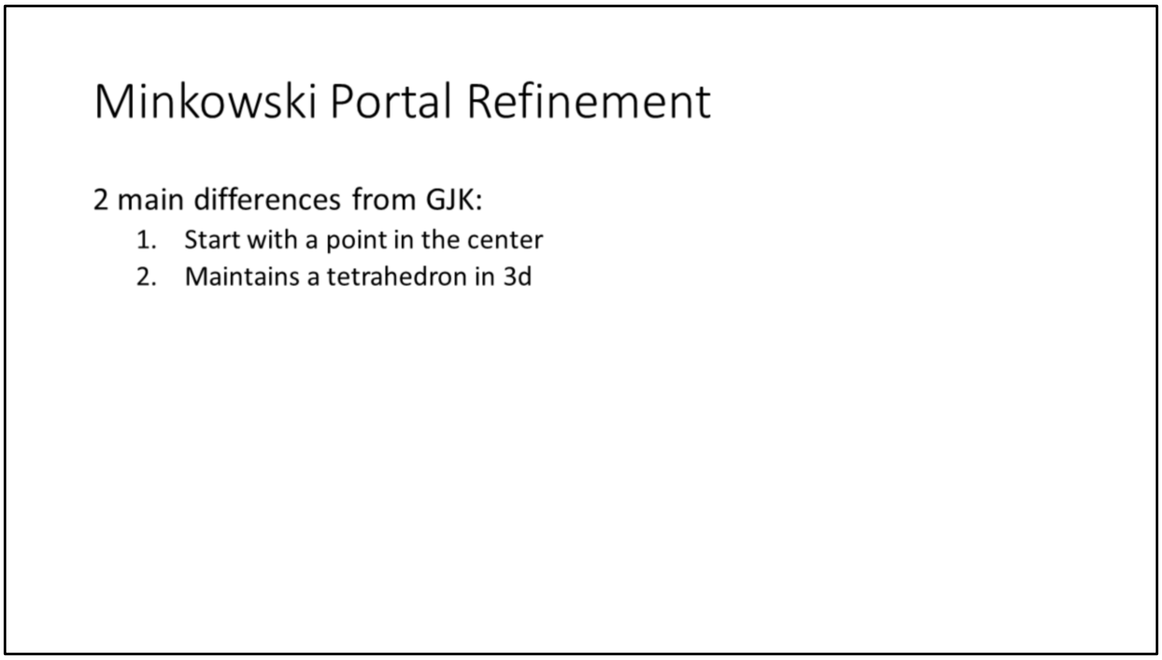


Minkowski Portal Refinement

Another support shape algorithm

Determines if a point is in a convex shape

Minkowski Portal Refinement (MPR) is another support shape algorithm like GJK. That is, it's an algorithm that tries to find if a point is in a convex hull using a support function. Like GJK, we can map convex shape vs. convex shape to point vs convex shape using Minkowski differences. As Minkowski differences have already been covered I will only talk about the actual MPR algorithm here: point vs. convex shape.



Minkowski Portal Refinement

2 main differences from GJK:

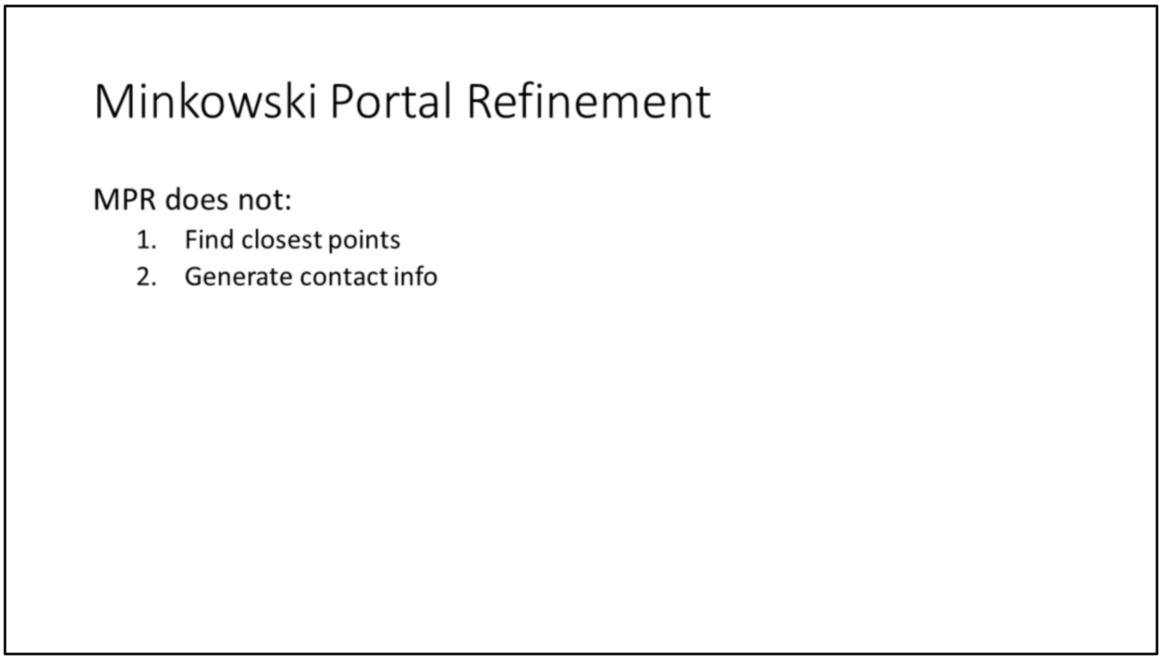
1. Start with a point in the center
2. Maintains a tetrahedron in 3d

There are 2 primary differences between GJK and MPR that cause the algorithms to be different.

First, GJK's points are always on the surface of the convex hull and all of them can appear/disappear throughout the course of the algorithm. Conversely, MPR starts with one point inside the convex hull and it never gets rid of that point.

Second, GJK's simplex can vary from 1 to 4 points at any point in the algorithm. MPR builds up to a tetrahedron (triangle in 2d) and keeps a tetrahedron for the rest of the algorithm.

The second point is the most important as it attempts to simplify the problem by having less cases to worry about.



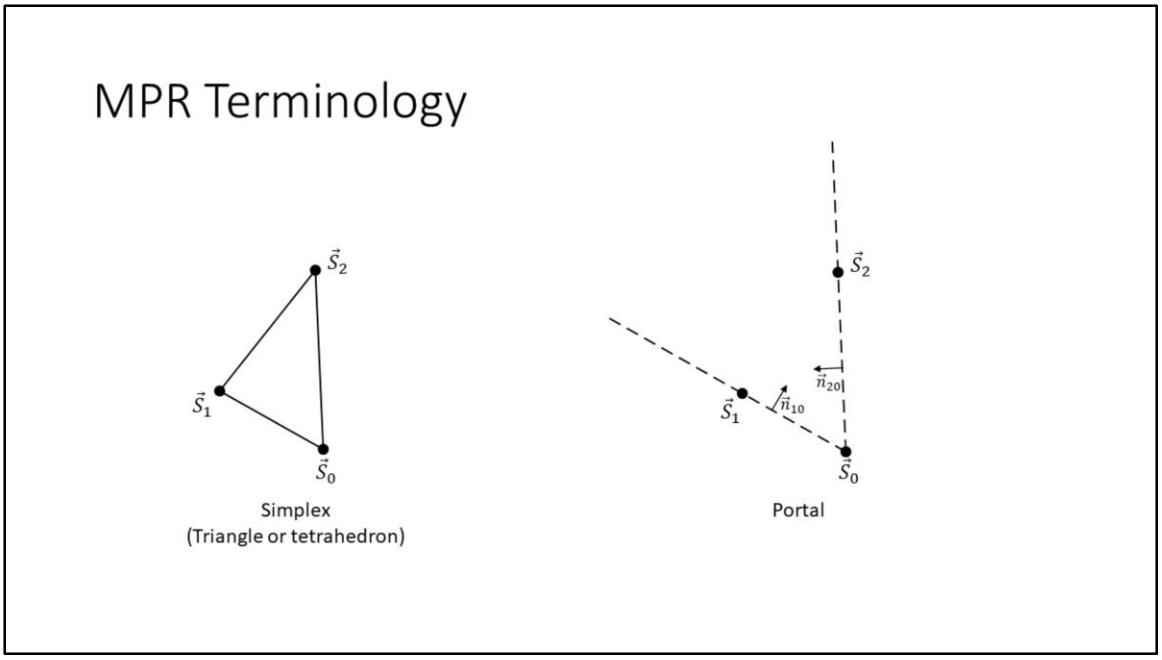
Minkowski Portal Refinement

MPR does not:

1. Find closest points
2. Generate contact info

There are a few things to point out about MPR. For the most part it obtains less information than GJK. GJK can find the closest point on the convex shape while MPR cannot. Also, GJK is unable to generate contact info which is also true for MPR.

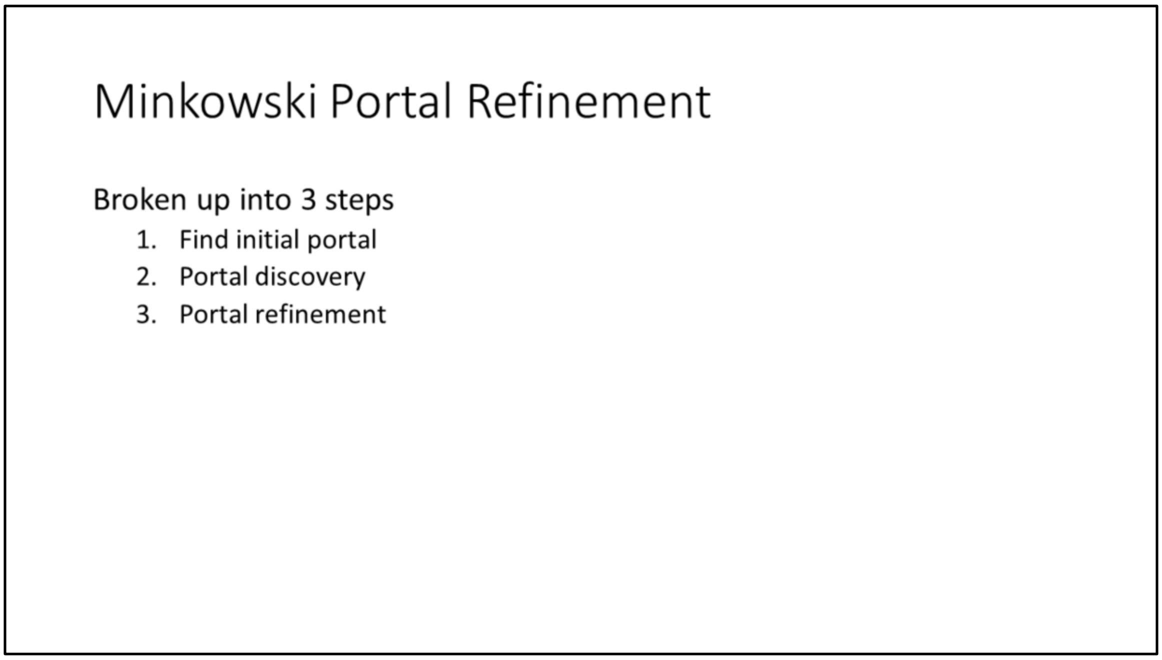
So why use MPR over GJK? Mainly due to geometric simplicity.



MPR Terminology

There's some quick terminology that needs to be covered first. In particular this concept of a portal.
The first term should be familiar: a simplex. In particular, we care about the simplex of our dimension (triangle in 2d, tetrahedron in 3d). As always, if we can find the query point $\vec{Q}$ in the simplex then we found containment.

The new concept is a portal. A portal is the infinite triangle (or tetrahedron in 3d) centered at $\vec{S}_0$; that is in 2d the portal is anything inside the positive region of $\vec{n}_{10}$ and $\vec{n}_{20}$. It is important to note that the portal is always centered at $\vec{S}_0$ as it is a point on the inside of the convex hull.



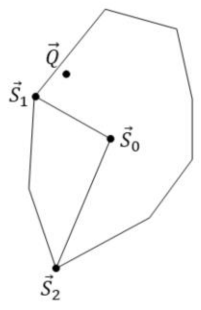
Minkowski Portal Refinement

Broken up into 3 steps

1. Find initial portal
2. Portal discovery
3. Portal refinement

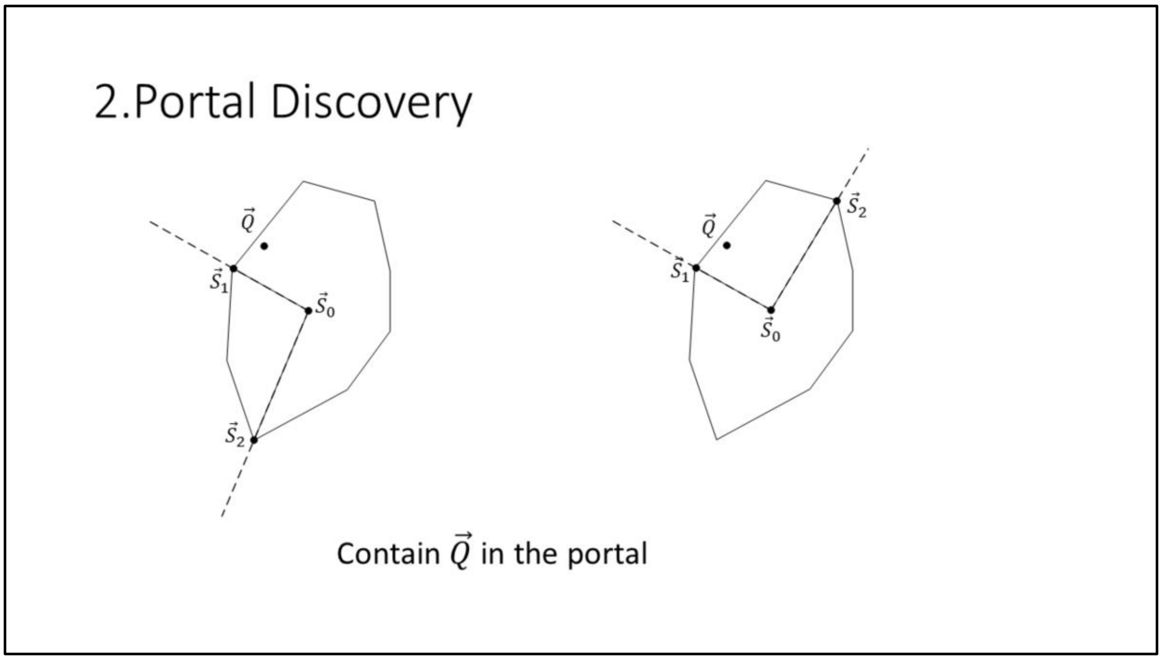
MPR is broken up into 3 main steps.

1.Find Initial Portal

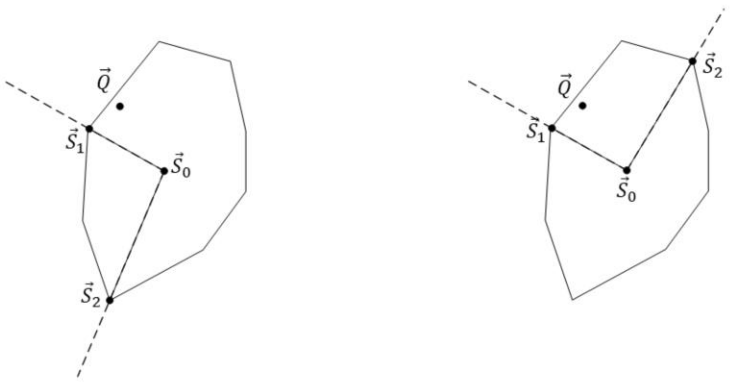


Build an initial portal

In the first step we just need to create the initial portal, that is build up to the correct number of points (3 in 2d and 4 in 3d). Technically how we do this doesn't matter, but a good initial portal choice can reduce other steps of the algorithm.

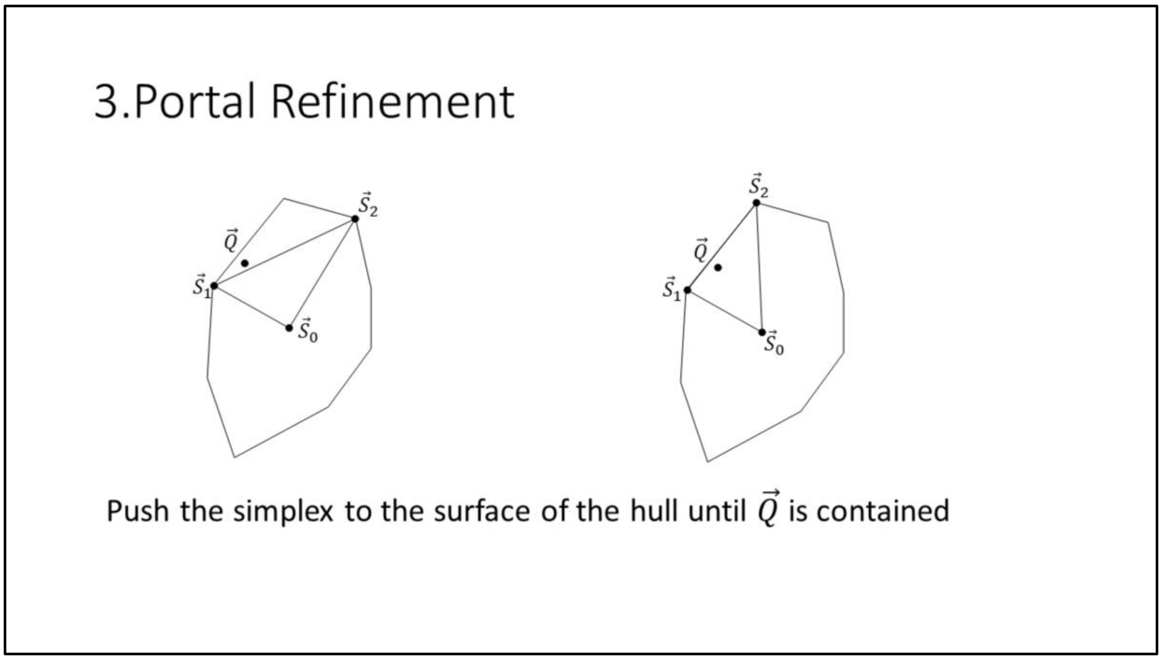


2.Portal Discovery



Contain $\vec{Q}$ in the portal

For the second part of the algorithm we want to get a portal that contains the origin. Note that this portal containing $\vec{Q}$ is different than the simplex containing $\vec{Q}$.



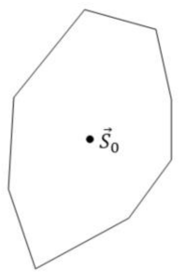
3.Portal Refinement

Push the simplex to the surface of the hull until $\vec{Q}$ is contained

The final step of the algorithm is to see if $\vec{Q}$ is actually contained in the hull. This is performed by trying to get the simplex to contain $\vec{Q}$ by pushing it to the surface. If we reach the surface and can't contain $\vec{Q}$ then $\vec{Q}$ is not in the hull.

Compute $\vec{S}_0$

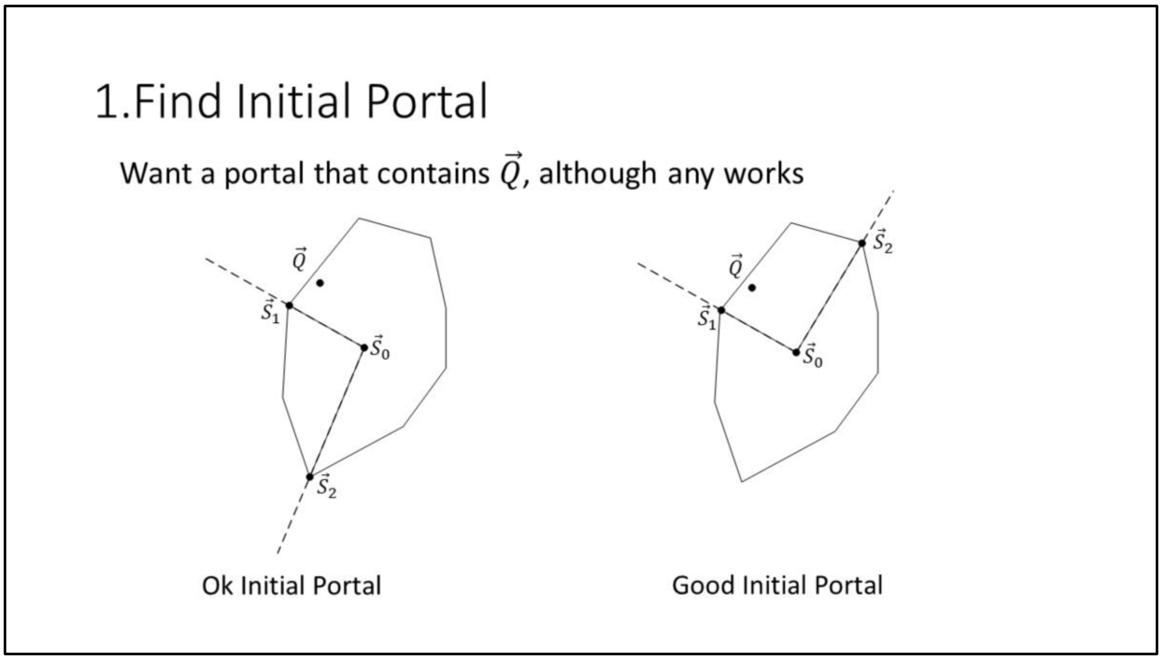
Compute any point inside the convex hull



For Minkowski difference choose $\vec{P}_B - \vec{P}_A$

That was the basic idea of the algorithm, now it’s time to describe in detail the actual steps.

To start with we need to compute a point inside the convex hull. Any deep point works. When using this for the Minkowski difference of two shapes a good choice is the difference of the two shape’s centers, that is $\vec{P}_B - \vec{P}_A$ where $\vec{P}$ is the centroid (or center of mass) of a shape.



1.Find Initial Portal

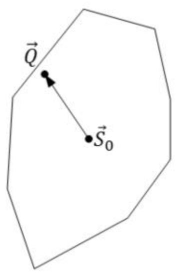
Want a portal that contains $\vec{Q}$, although any works

Finding the initial portal can be done pretty much any way that creates a non-degenerate portal, but some initial portals are better than others. If we can construct the initial portal so it is more likely to contain $\vec{Q}$ then we'll reduce the time of the next step.

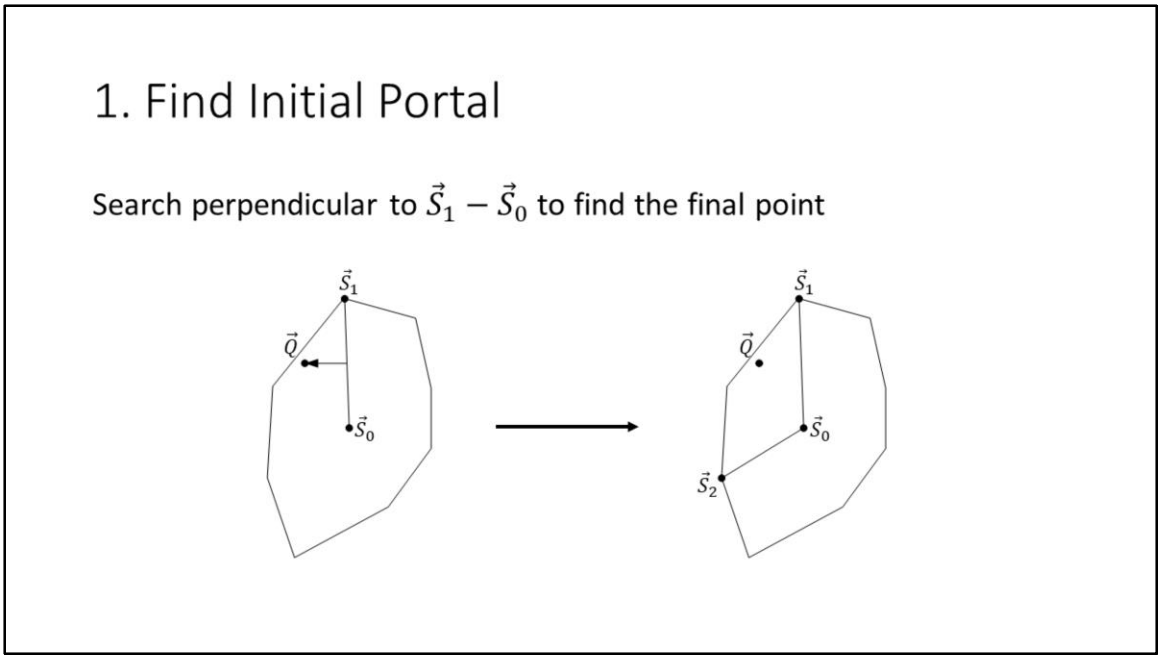
1. Find Initial Portal

We only have $\vec{Q}$ and $\vec{S}_0$

Search $\vec{d} = \vec{Q} - \vec{S}_0$ for a new point

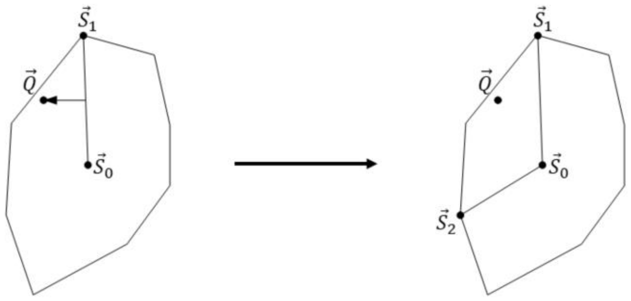


Now we only have 2 pieces of information: $\vec{Q}$ and $\vec{S}_0$. We can search now in the direction $\vec{Q} - \vec{S}_0$ to find a new point.

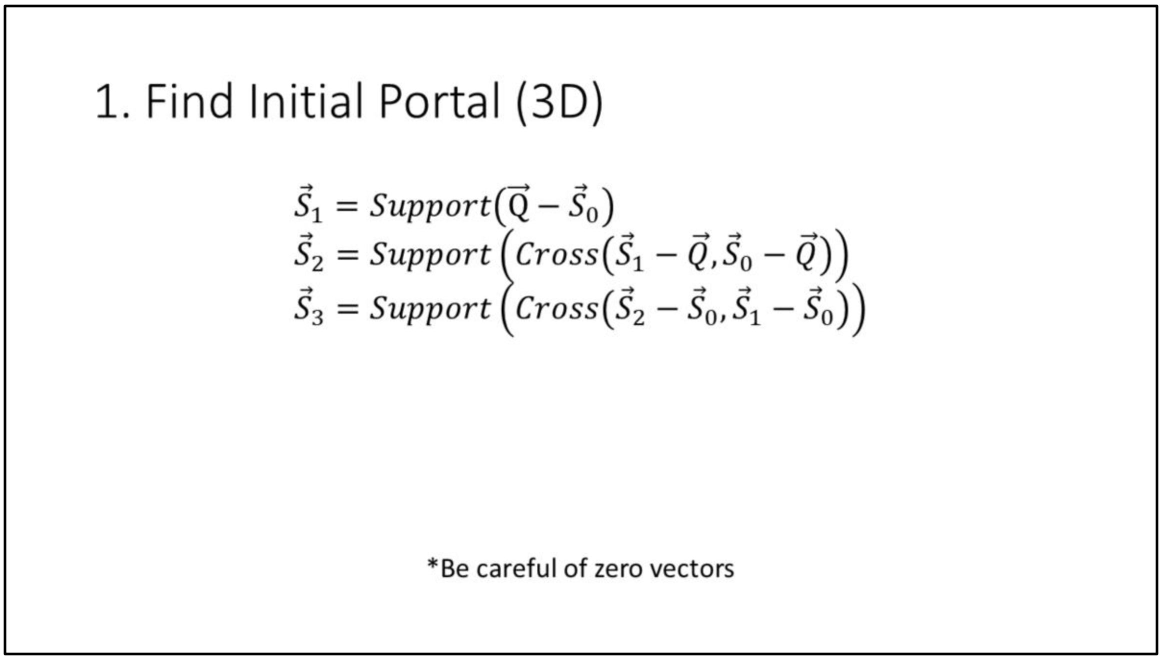


1. Find Initial Portal

Search perpendicular to $\vec{S}_1 - \vec{S}_0$ to find the final point



In 2d the final point can then be found by search in a direction perpendicular to $\vec{S}_1 - \vec{S}_0$. To make this slightly better we can make sure that the search direction is towards $\vec{Q}$ and in so doing in 2d we'll always contain $\vec{Q}$ in the initial portal and we can skip step 2 of the algorithm.



1. Find Initial Portal (3D)

$$\vec{S}_1 = Support(\vec{\mathrm{Q}} - \vec{S}_0)$$
$$\vec{S}_2 = Support\left(Cross(\vec{S}_1 - \vec{Q}, \vec{S}_0 - \vec{Q})\right)$$
$$\vec{S}_3 = Support\left(Cross(\vec{S}_2 - \vec{S}_0, \vec{S}_1 - \vec{S}_0)\right)$$

*Be careful of zero vectors

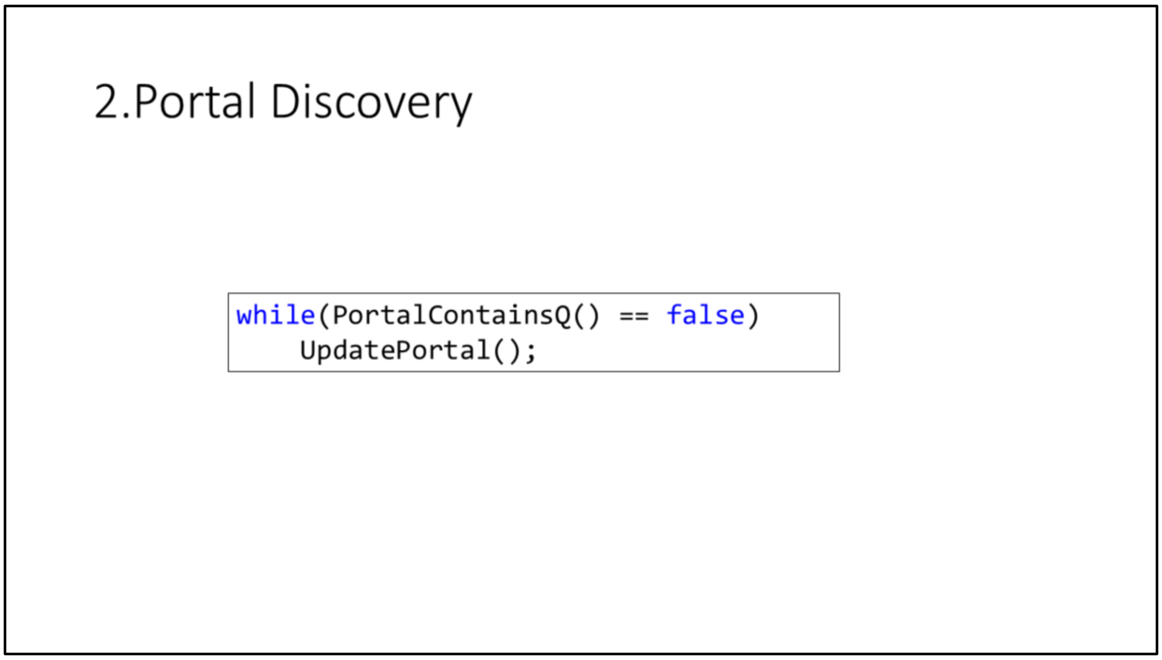
In 3d finding the initial portal is a bit trickier.

We can start the exact same with $\vec{S}_1$ as before.

For $\vec{S}_2$ we want the same conceptual thing: a vector perpendicular to $\vec{S}_1 - \vec{S}_0$, however there are infinitely many now. Instead we assume that $\vec{S}_1$ will not be in the same exact direction as our first support direction. Hence we can instead compute a vector perpendicular to the plane containing $\vec{S}_0$, $\vec{S}_1$, and $\vec{Q}$. Do not that this case can (and will) produce zero vectors. If that is the case then we can either pick a random vector.

Finally we want to get the final point of the portal, ideally to contain $\vec{Q}$. Once again we choose a vector perpendicular to the plane containing $\vec{S}_0$, $\vec{S}_1$, and $\vec{S}_2$. However, by choosing the cross product order carefully we're guaranteed that this direction will towards $\vec{Q}$.

Note though that in 3d we can make no guarantee that this step will force $\vec{Q}$ to be contained within the portal.



2.Portal Discovery

```
while(PortalContainsQ() == false)
    UpdatePortal();
```

Now that we have a portal we need to make sure the origin is contained within it.

This portion of the algorithm is contained of just 2 steps. As long as the portal doesn't contain $\vec{Q}$ then we update it by removing 1 point and adding another.

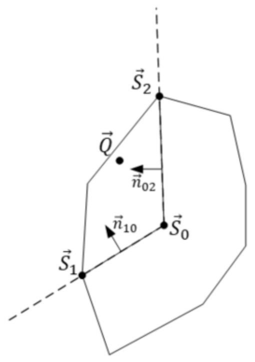
One thing to note about this algorithm going forward, we can always construct vectors that are guaranteed to point the direction we want with a bit of extra work, but if we properly maintain winding order we can avoid extra computations.
Also note that from here on out I'm only going to talk about 3d as 2d is a simplification of that (except for pictures).

Portal Contains Q

Check if $\vec{Q}$ is inside all faces

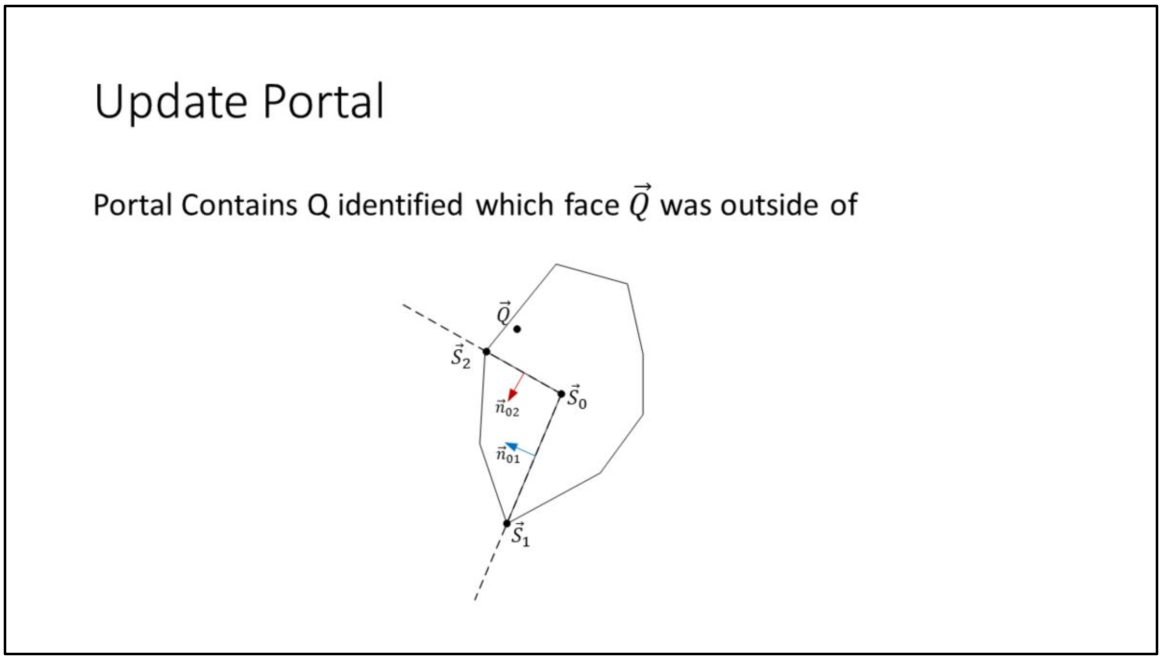
Can verify $\vec{n}$ direction from other point

Or maintain consistent winding order



To determine if the portal contains $\vec{Q}$ we have to see if $\vec{Q}$ is on the positive side of all of the portal faces. In 2d this is the faces $\vec{S}_0\vec{S}_1$ and $\vec{S}_2\vec{S}_0$. To compute the clockwise normal for a vector in 2d we can just temporarily convert to 3d and compute $Cross(Vector3(v.x, v.y, 0), \vec{z}))$ and then take the resultant x and y components back to 2d. So to compute $\vec{n}_{10}$ we compute the clockwise normal from the vector $\vec{v} = \vec{S}_1 - \vec{S}_0$.

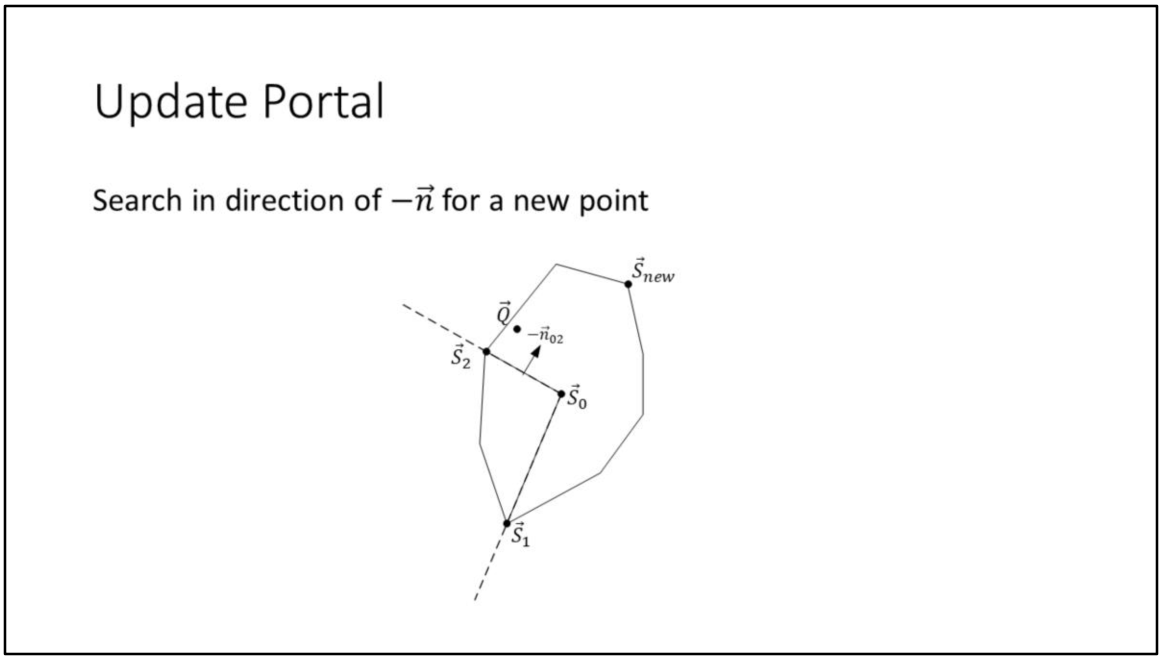
Similar operations are performed in 3d, only we have 3 faces: $\vec{S}_0\vec{S}_1\vec{S}_2$, $\vec{S}_0\vec{S}_2\vec{S}_3$, and $\vec{S}_0\vec{S}_3\vec{S}_1$. In this case the normal is defined for face $\vec{S}_0\vec{S}_1\vec{S}_2$ as $\vec{n}_{012} = Cross(\vec{S}_2 -$



Update Portal

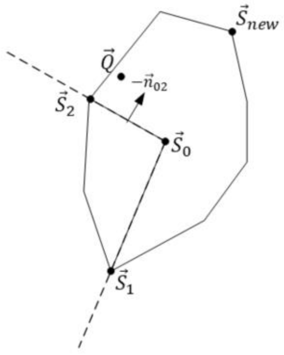
Portal Contains Q identified which face $\vec{Q}$ was outside of

If we make it to the Update Portal step then we know $\vec{Q}$ is outside one of the faces. Without loss of generality we'll assume that $\vec{Q}$ is outside the face $\vec{S}_0\vec{S}_2$. We know that the point not on the face ($\vec{S}_1$) isn't contributing so we need to find a new point to replace it with.

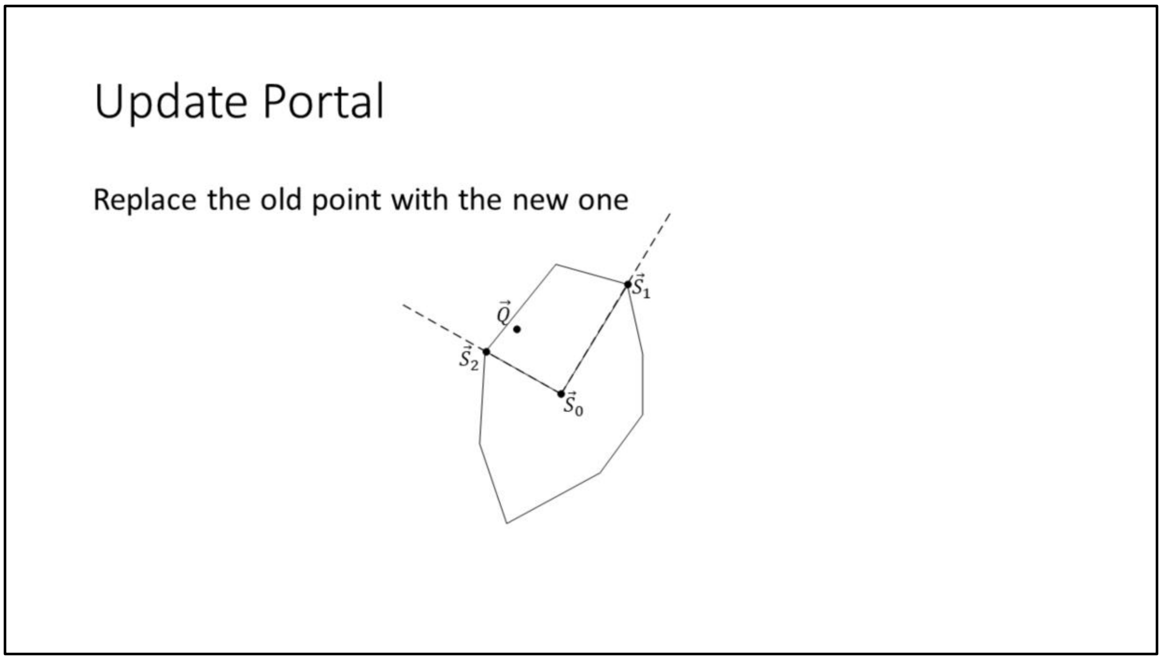


Update Portal

Search in direction of $-\vec{n}$ for a new point

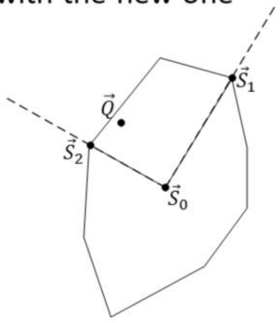


To replace $\vec{S}_1$ we should look in the opposite direction of the normal whose face we’re outside of. In this case we’d look in the direction $-\vec{n}_{02}$ since $\vec{Q}$ is outside of the face $\vec{S}_0\vec{S}_2$. When we do this search we’ll get a new point that we’ll label for now $\vec{S}_{new}$.

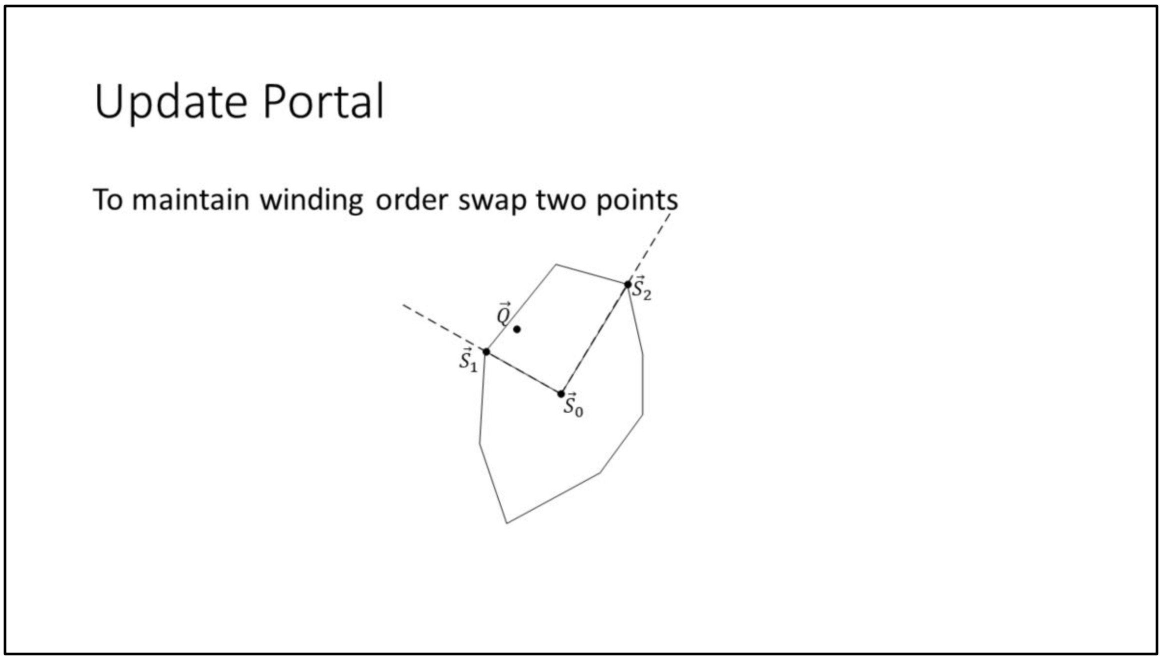


Update Portal

Replace the old point with the new one

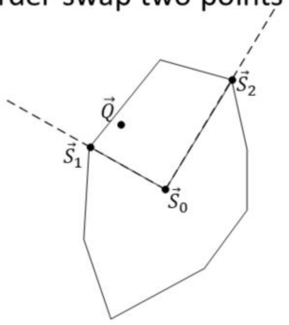


Since our search direction was from face $\vec{S}_0\vec{S}_2$ we know that the point that wasn't a part of this face is no longer relevant, that is the old point $\vec{S}_1$. So we can replace $\vec{S}_1$ with the point we just found, $\vec{S}_{new}$.



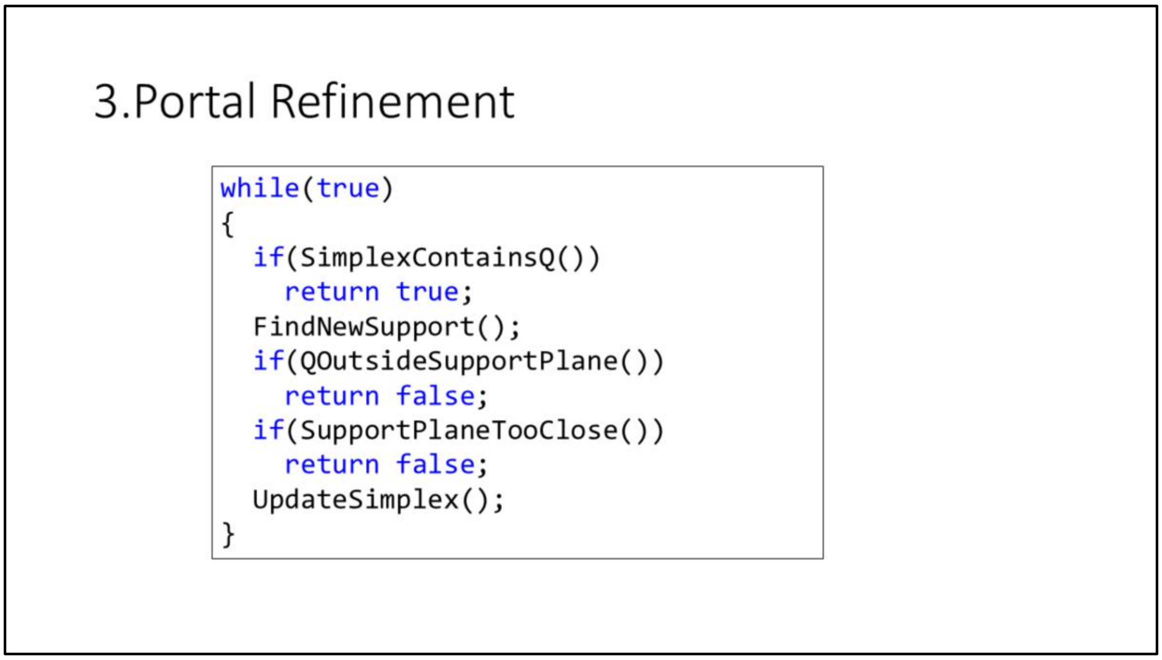
Update Portal

To maintain winding order swap two points



The only problem with this is that by definition we've flipped the winding order of our simplex (in 2d $\vec{Q}$ should always be clockwise of $\vec{S}_0\vec{S}_1$ and counter clockwise of $\vec{S}_0\vec{S}_2$). This is easy to fix though as we just have to flip two points every time; that is swap points $\vec{S}_1$ and $\vec{S}_2$ after every point replacement.

This also holds true in 3d. The winding order of the portal face $\vec{S}_1\vec{S}_2\vec{S}_3$ needs to be maintained. Whenever the portal is updated the order will be flipped. To fix this any 2 points on the portal face need to be swapped.



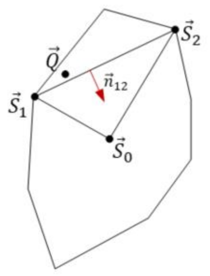
3.Portal Refinement

```
while(true)
{
  if(SimplexContainsQ())
    return true;
  FindNewSupport();
  if(QOutsideSupportPlane())
    return false;
  if(SupportPlaneTooClose())
    return false;
  UpdateSimplex();
}
```

Now for the final portion, the portal refinement. Remember the idea here is to push to surface of the portal out towards the surface of the CSO until it contains $\vec{Q}$. Presented here are the steps for the algorithm, although keep in mind it is often good to replace the while(true) with some max iterations counter.

Simplex Contains Q

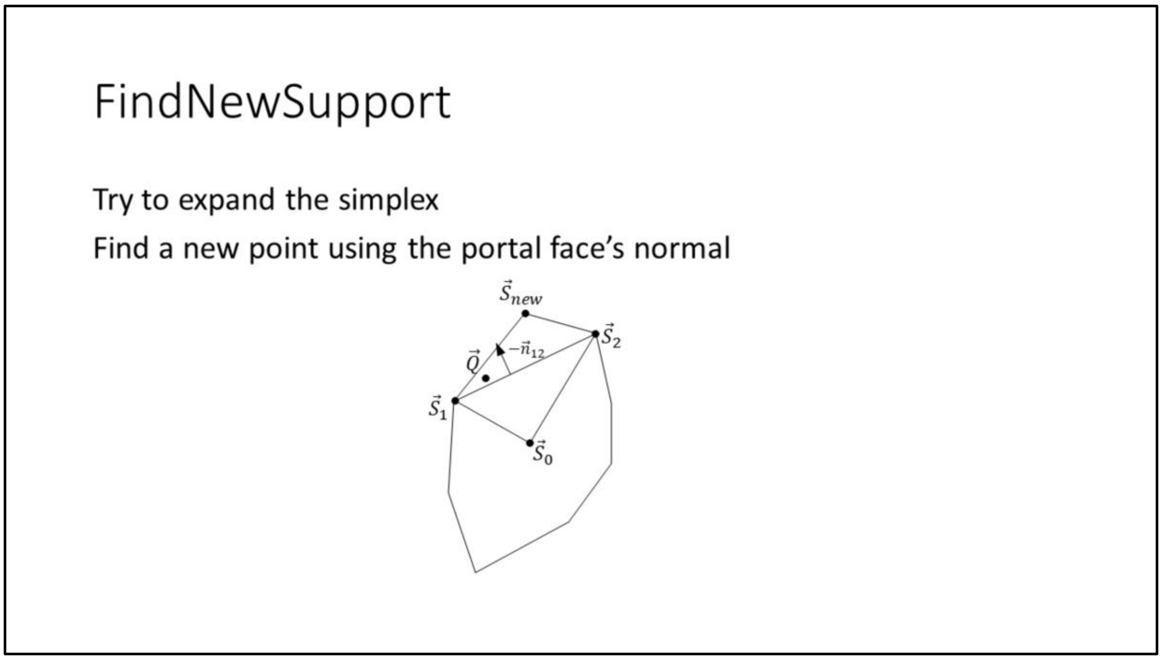
Check if $\vec{Q}$ is inside the portal face



Our primary goal is to contain $\vec{Q}$ within the simplex. If we can do this we know that $\vec{Q}$ is inside of the convex shape.

The naive way to do this is to check if $\vec{Q}$ is inside all of the simplex faces (3 for 2d, 4 for 3d), however the Portal Discovery step guaranteed that $\vec{Q}$ was within all faces except for the portal's face, i.e. we know $\vec{Q}$ is within $\vec{S}_{01}$ and $\vec{S}_{20}$. Hence we'll know if $\vec{Q}$ is within the simplex if we just check the final face. Note that we'll maintain this guarantee when updating the simplex.

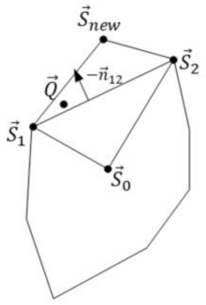
In 2d we can verify this by just checking the face normal of $\vec{S}_1\vec{S}_2$ and in 3d we can check the normal of $\vec{S}_1\vec{S}_3\vec{S}_2$. If $\vec{Q}$ is on the positive side of this face then $\vec{Q}$ is within the hull and the algorithm returns true. Otherwise we only know that $\vec{Q}$ isn't in the simplex, but it could still be in the hull. We need to further expand the simplex towards the surface of the hull.



FindNewSupport

Try to expand the simplex

Find a new point using the portal face's normal

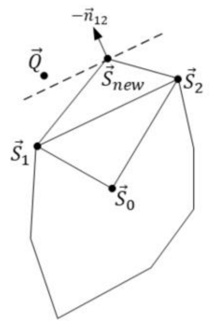


If $\vec{Q}$ is not inside the simplex then we need to push the portal towards the surface. To do this we need to find a new point that is within the current portal and closer towards the surface. The best search direction for this is to use the outward normal of the portal face, in this case $-\vec{n}_{12}$.

Before updating the simplex to include $\vec{S}_{new}$ there's a number of termination conditions that need to be tested.

Q Outside Support Plane

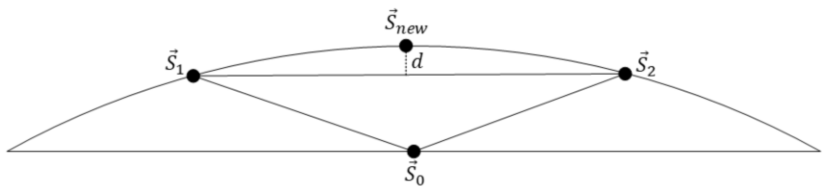
Check if containing $\vec{Q}$ is even possible



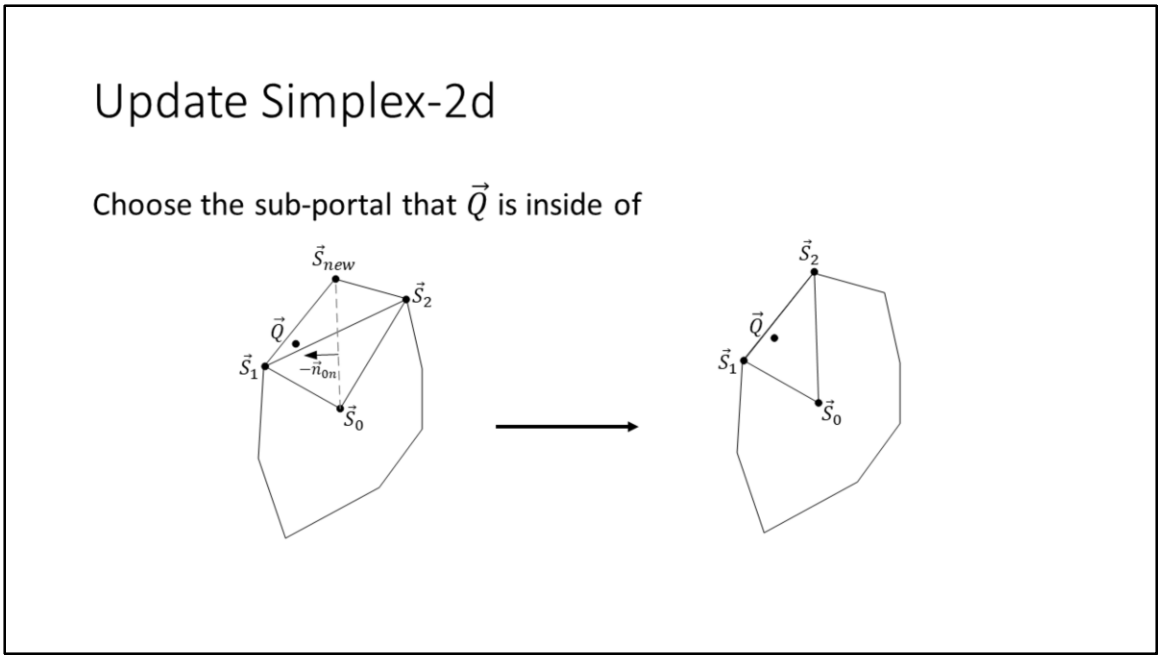
As we've now found the point furthest in the direction of $\vec{S}_{123}$ we can check to see if it's even possible to contain $\vec{Q}$. If $\vec{S}_{new}$ is not further that $\vec{Q}$ in our search direction then there's no way we can ever contain it. In this case the algorithm terminates with false.

Support Plane Too Close

Check if $d < \epsilon$

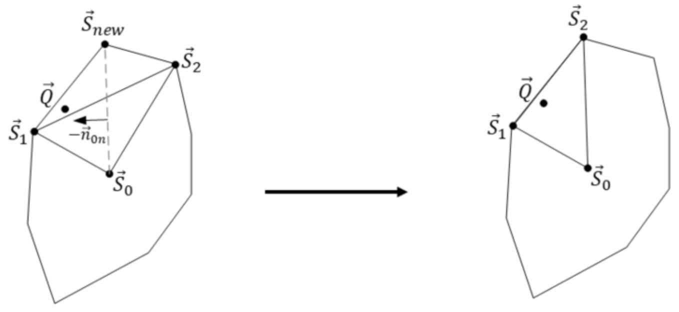


The other stopping condition here is to see if we've made it to the surface of the CSO (or close enough). If $\vec{S}_{new}$ is on the plane or the portal face then we've pushed all the way to the surface and we still couldn't contain $\vec{Q}$, hence we're done. To properly deal with analytic shapes (and just numerical robustness in general), this is supplemented with an epsilon check.



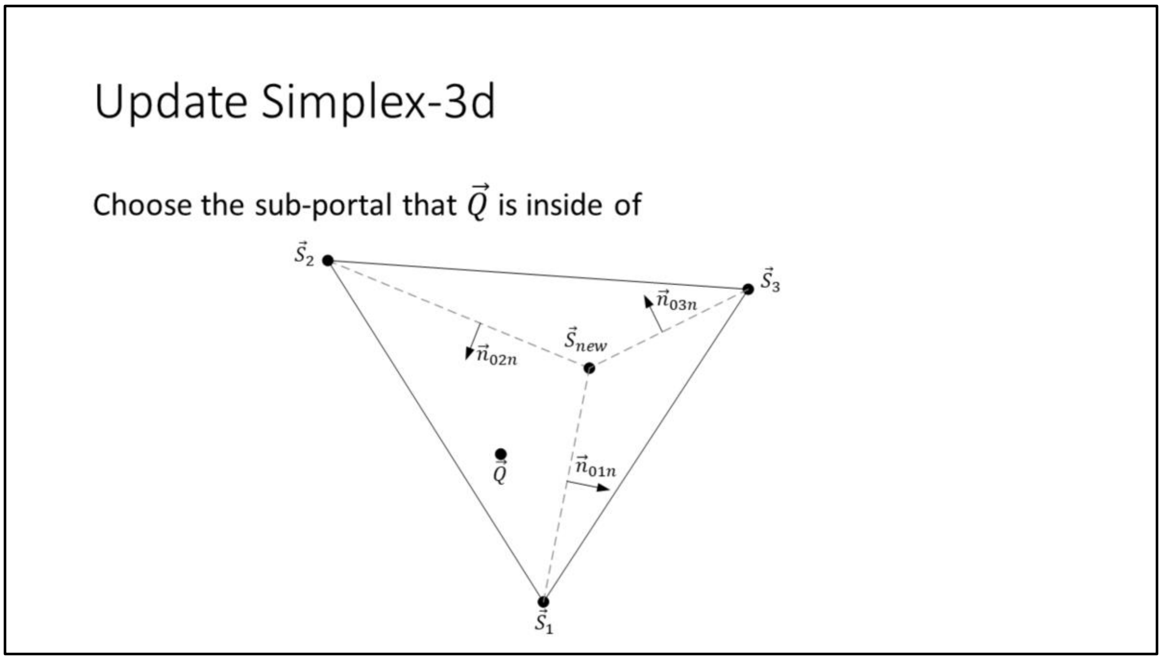
Update Simplex-2d

Choose the sub-portal that $\vec{Q}$ is inside of



With our termination conditions out of the way we need to update the simplex to be closer to containing $\vec{Q}$ using $\vec{S}_{new}$. As $\vec{S}_{new}$ is guaranteed to be in-between $\vec{S}_1$ and $\vec{S}_2$ we can think of the vector from $\vec{S}_0$ to $\vec{S}_{new}$ as sub-dividing the portal into 2 sub-portals. We need to choose one of these sub-portals to as our new simplex. Which one we choose is based upon which one contains $\vec{Q}$.

To determine which sub-port $\vec{Q}$ is in we just have to check the normal of the new face: $\vec{n}_{0n}$ (note the n sub-script refers here to the point $\vec{S}_{new}$, not the normal). Once we know which side $\vec{Q}$ is on we can replace the point not part of the sub-portal with $\vec{S}_{new}$. Note that the winding order doesn't change here.



Update Simplex-3d

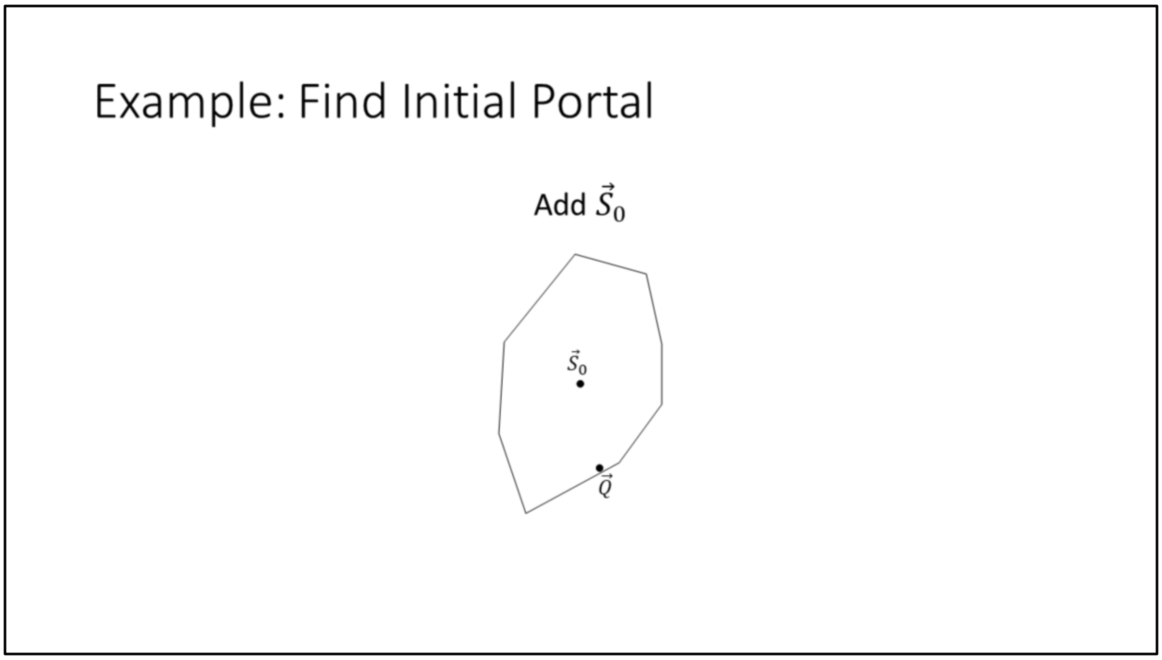
Choose the sub-portal that $\vec{Q}$ is inside of

Updating the simplex in 3d is a little trickier. Instead of sub-dividing the portal into 2 sub-portals, we get 3. To determine which sub-portal we're in we have to check 2 planes. In this case $\vec{Q}$ is within the sub-portal of $\vec{S}_1\vec{S}_2\vec{S}_{new}$ which we can verify by seeing that it's on the front side of $\vec{n}_{02n}$ and on the back side of $\vec{n}_{01n}$.

Likewise if $\vec{Q}$ was in the sub-portal of $\vec{S}_1\vec{S}_{new}\vec{S}_3$ then it would be on the positive side of $\vec{n}_{01n}$ while being on the back side of $\vec{n}_{03n}$.

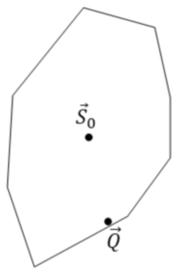
Like before, once we've identified which sub-portal $\vec{Q}$ is indide of we just replace the point from the simplex that isn't part of the sub-portal with $\vec{S}_{new}$. In this example $\vec{S}_3$ is replaced.

Note that I didn't draw in $\vec{S}_0$ to avoid cluttering the picture.



Example: Find Initial Portal

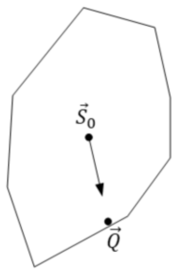
Add $\vec{S}_0$



The first step is to just add the first point of the simplex: $\vec{S}_0$. I just chose an arbitrary point in the center.

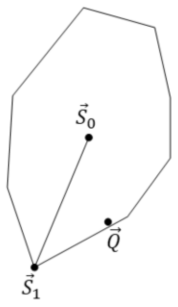
Example: Find Initial Portal

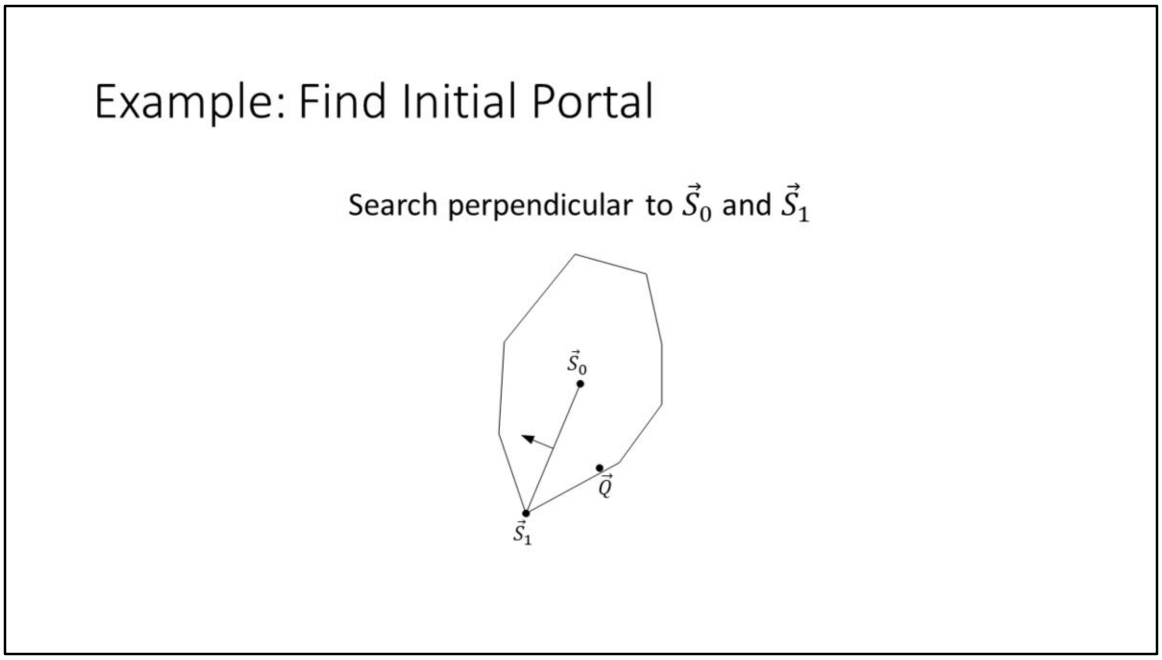
Search in direction $\vec{Q} - \vec{S}_0$



Example: Find Initial Portal

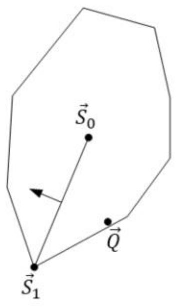
Add $\vec{S}_1$ to the simplex





Example: Find Initial Portal

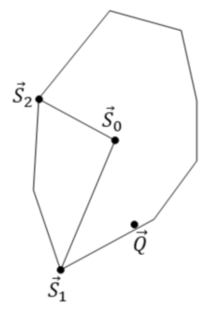
Search perpendicular to $\vec{S}_0$ and $\vec{S}_1$

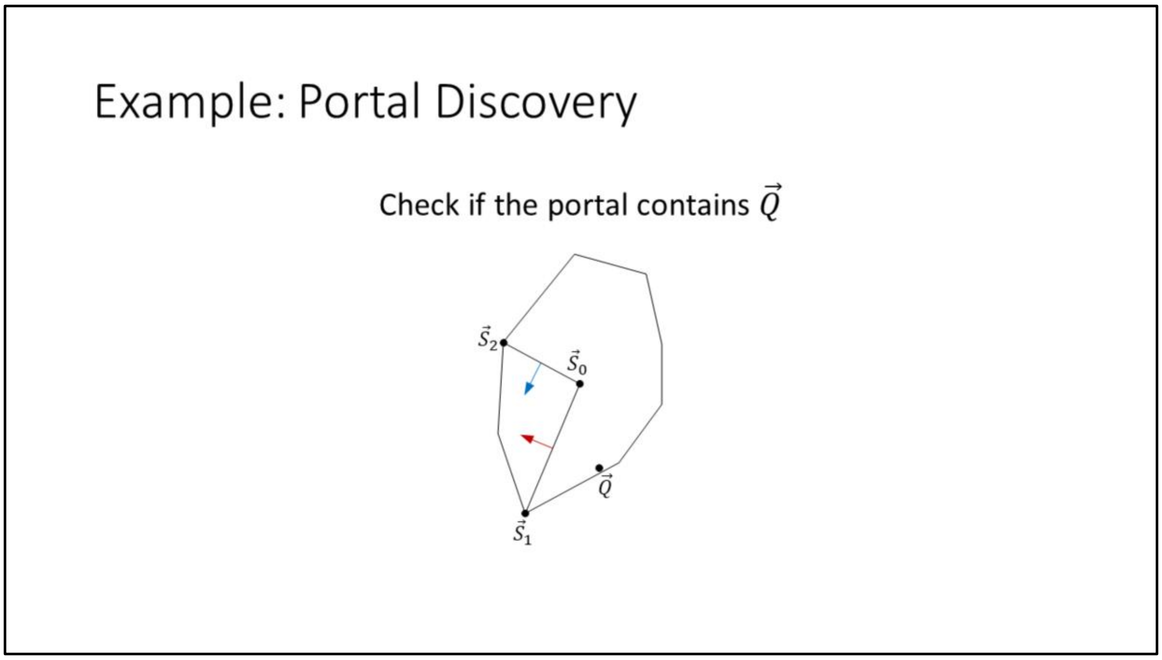


Now we need a vector perpendicular to $\vec{S}_0$ and $\vec{S}_1$. Ideally we'd choose to search in the opposite direction but to illustrate the portal discovery step properly I chose to search in the opposite direction.

Example: Find Initial Portal

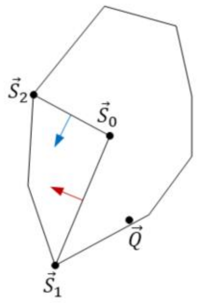
Add $\vec{S}_2$



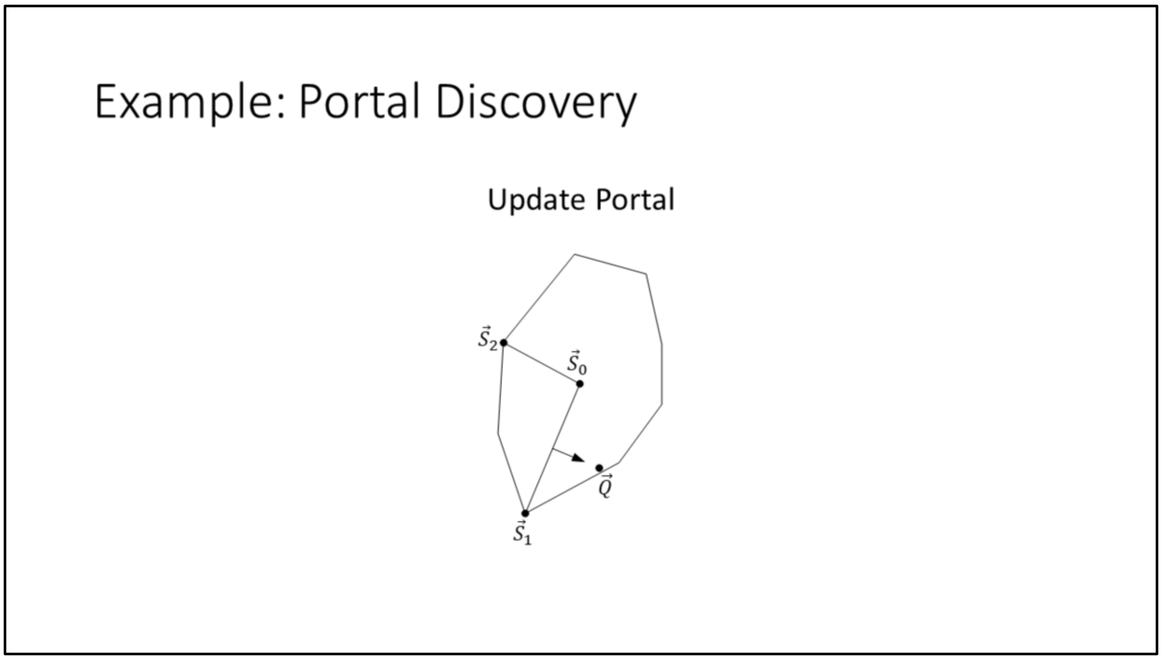


Example: Portal Discovery

Check if the portal contains $\vec{Q}$

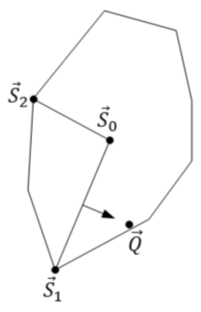


Now we've entered the portal discovery step. We want to see if the portal contains $\vec{Q}$. We do this by checking the normals of each face. In this example $\vec{Q}$ is on the positive side of $\vec{S}_2\vec{S}_0$ but on the negative side of $\vec{S}_1\vec{S}_0$.

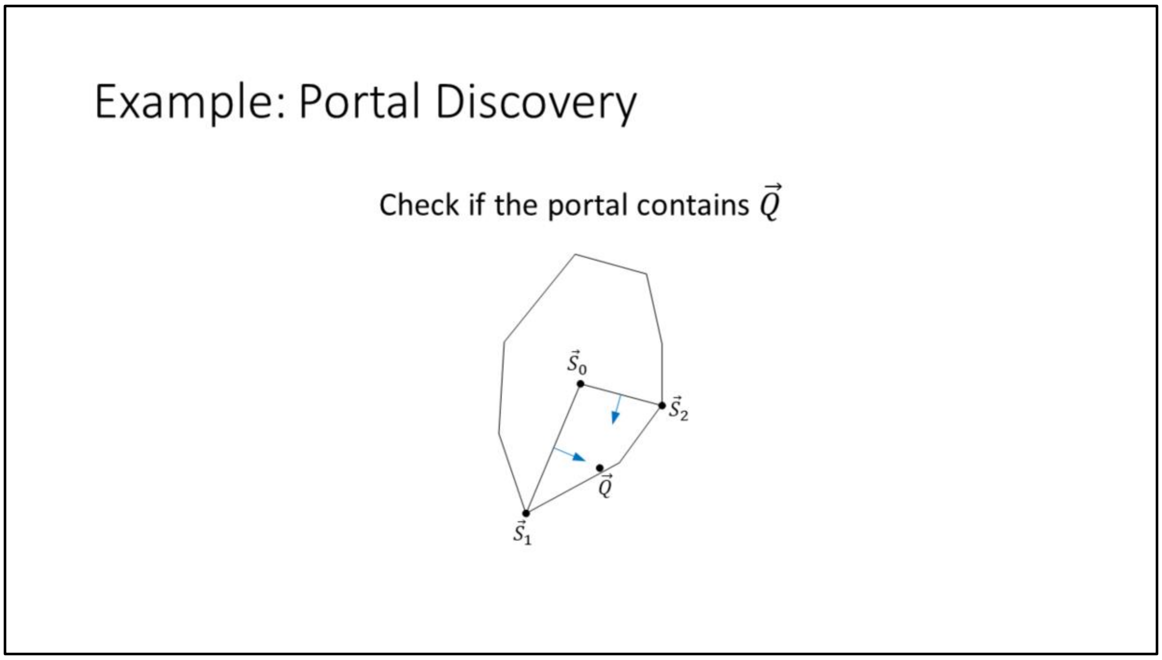


Example: Portal Discovery

Update Portal

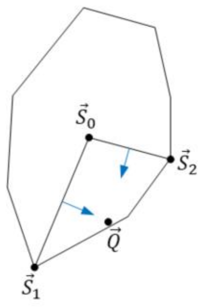


Since $\vec{Q}$ was on the outside of $\vec{S}_1\vec{S}_0$ we search in the opposite direction of its normal for a new point.

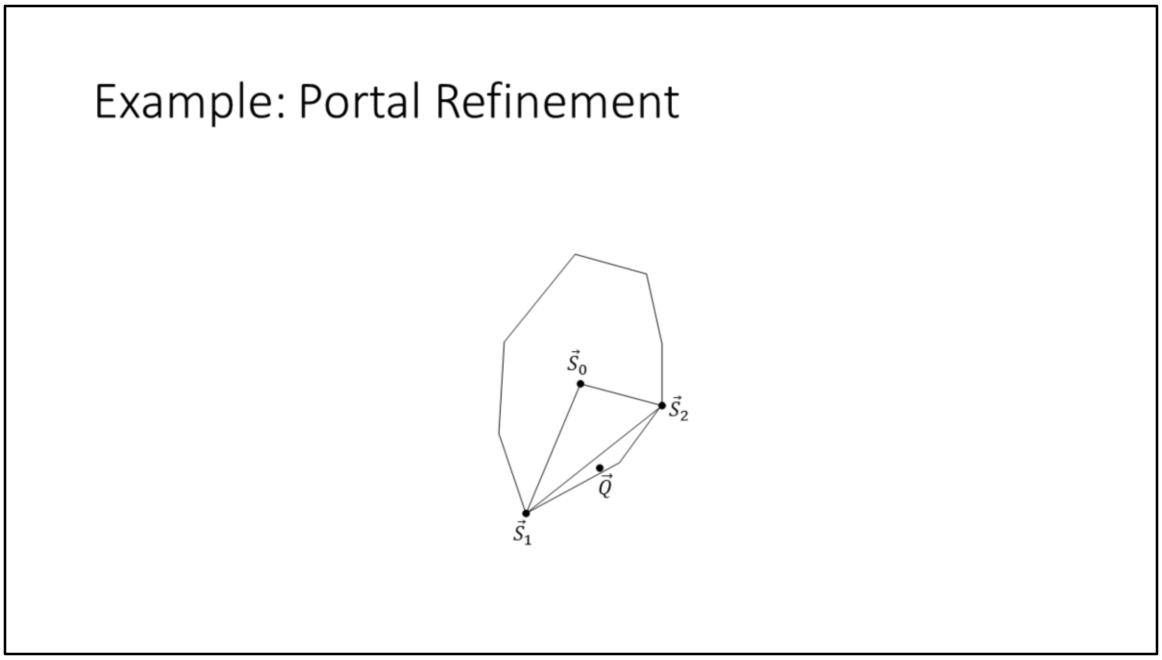


Example: Portal Discovery

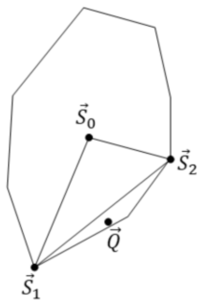
Check if the portal contains $\vec{Q}$



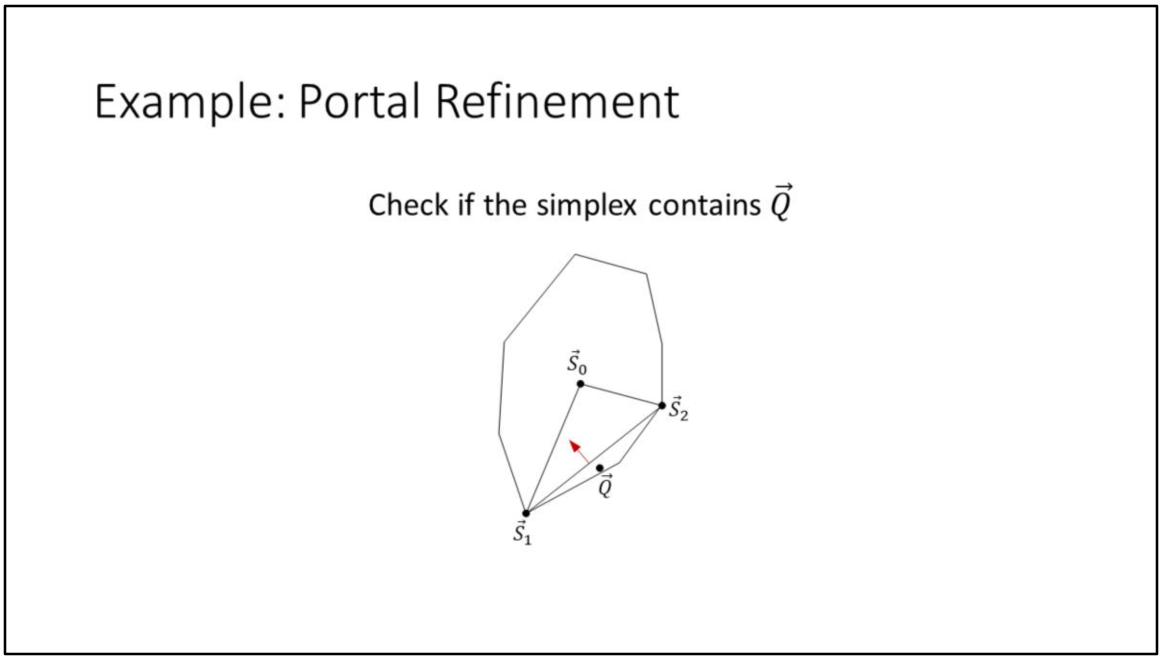
Once again check if the portal contains $\vec{Q}$. This time it does so we can move onto portal refinement.



Example: Portal Refinement

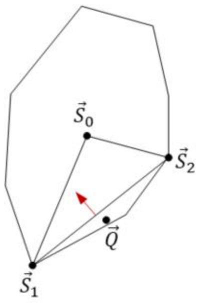


Now our goal is to push the simplex out untill it contains $\vec{Q}$ or we can't expand it anymore.



Example: Portal Refinement

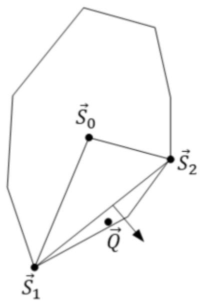
Check if the simplex contains $\vec{Q}$



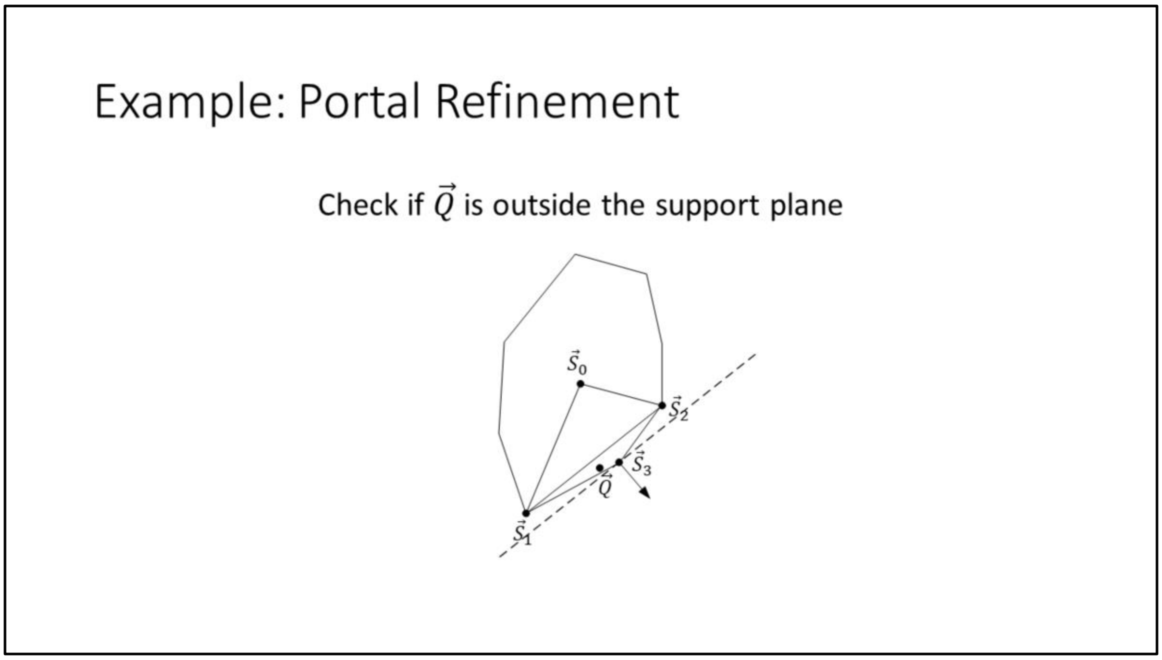
Remember to check if the simplex contains $\vec{Q}$ we only have to see if it's on the inside of the portal face $\vec{S}_1\vec{S}_2$ as we know it's within the infinite portal.
In this case it is not inside the simplex, but we don't know if it's outside the shape so we have to attempt to push the simplex towards the surface.

Example: Portal Refinement

Find a new support

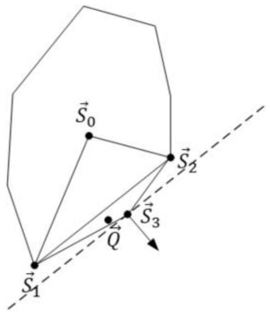


To expand the simplex we search in the direction of the portal face $\vec{S}_1\vec{S}_2$.



Example: Portal Refinement

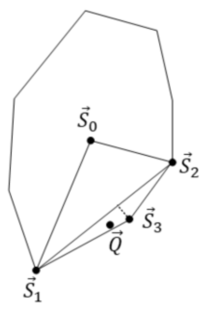
Check if $\vec{Q}$ is outside the support plane



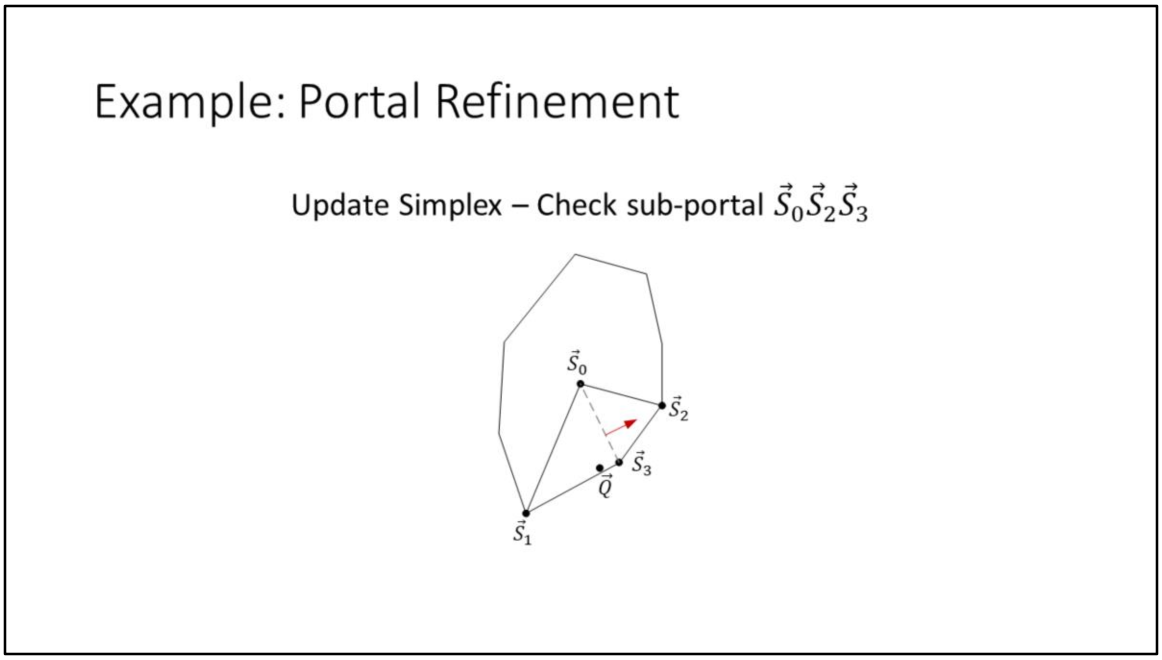
If $\vec{Q}$ is outside the plane defined by the normal of the portal and the new support point then we can't ever contain $\vec{Q}$ in the shape and hence we'd terminate.

Example: Portal Refinement

Check if the support plane is too close

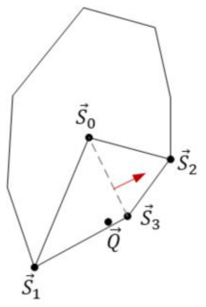


For numerical robustness we check if the new simplex point $\vec{S}_3$ is close enough to the old portal face. If it was we assume we can’t make enough progress forward and so we terminate.

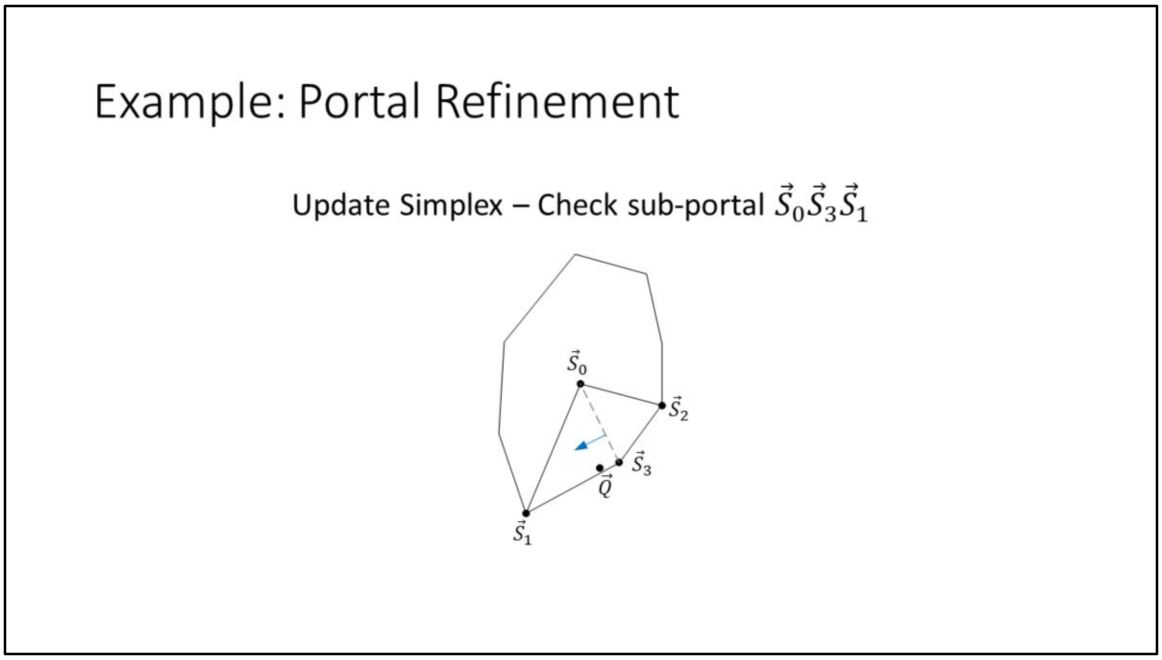


Example: Portal Refinement

Update Simplex – Check sub-portal $\vec{S}_0\vec{S}_2\vec{S}_3$

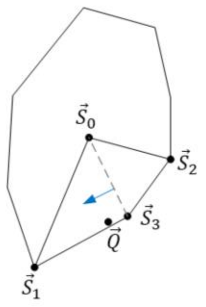


To update the simplex we need to test each sub-portal to see which one $\vec{Q}$ is within. We do this by checking the division formed by the new point $\vec{S}_3$. In this case $\vec{Q}$ is not in the sub-portal $\vec{S}_0\vec{S}_2\vec{S}_3$.

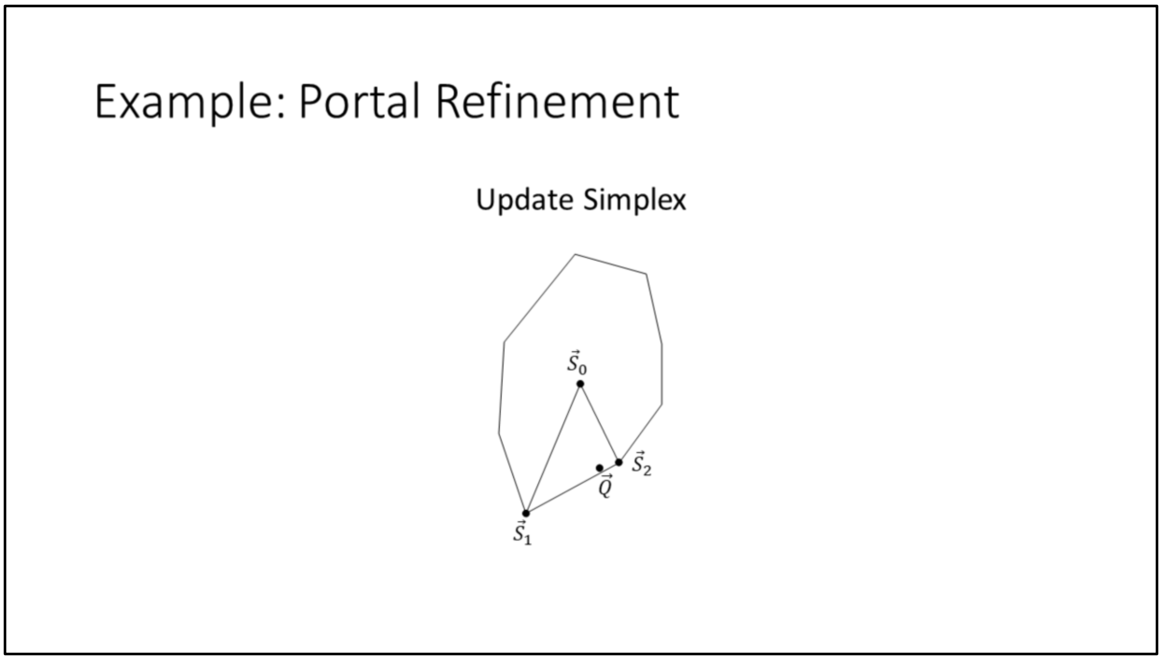


Example: Portal Refinement

Update Simplex – Check sub-portal $\vec{S}_0\vec{S}_3\vec{S}_1$

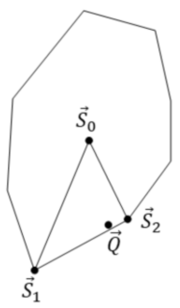


Now we check the sub-portal $\vec{S}_0\vec{S}_3\vec{S}_1$ and find that $\vec{Q}$ is inside.

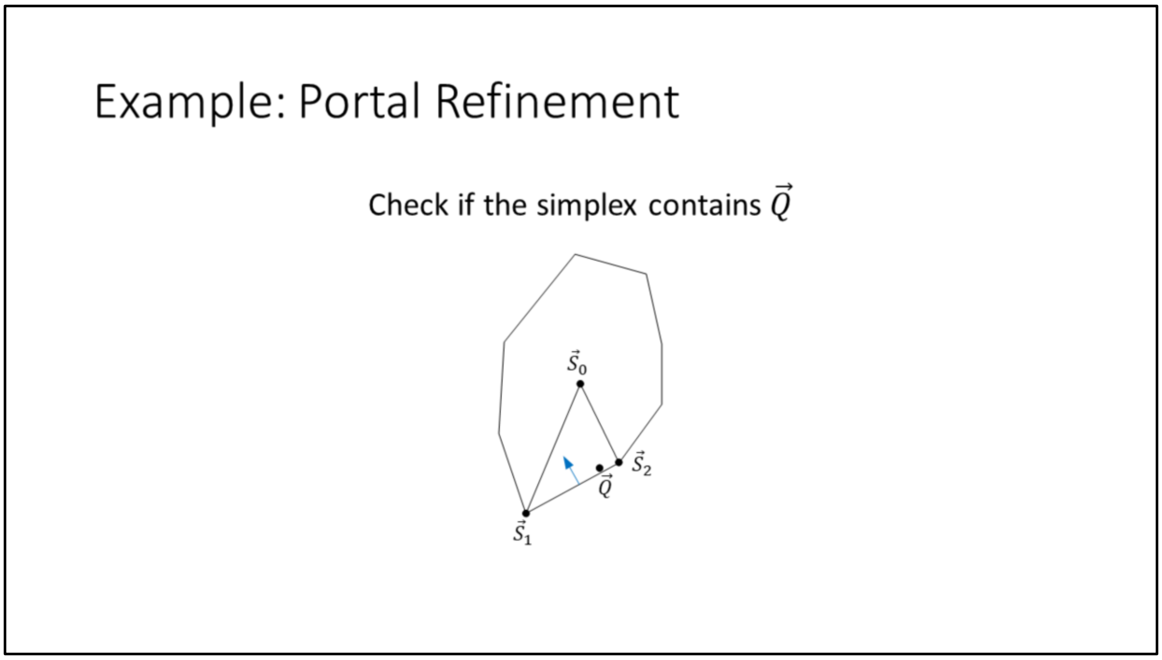


Example: Portal Refinement

Update Simplex

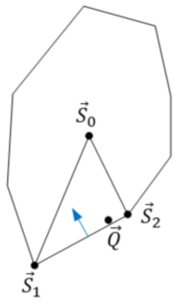


Since $\vec{Q}$ was inside the sub-portal $\vec{S}_0\vec{S}_3\vec{S}_1$ we replace the un-used point $\vec{S}_2$ with the support point we found $\vec{S}_3$.



Example: Portal Refinement

Check if the simplex contains $\vec{Q}$



Now the portal refinement starts over. We check if the portal face contains $\vec{Q}$. In this case it does so we know the hull contains $\vec{Q}$.

Questions?