Pathfinding with Navigation Meshes

Filipe Ghesla Silvestrim

Outline

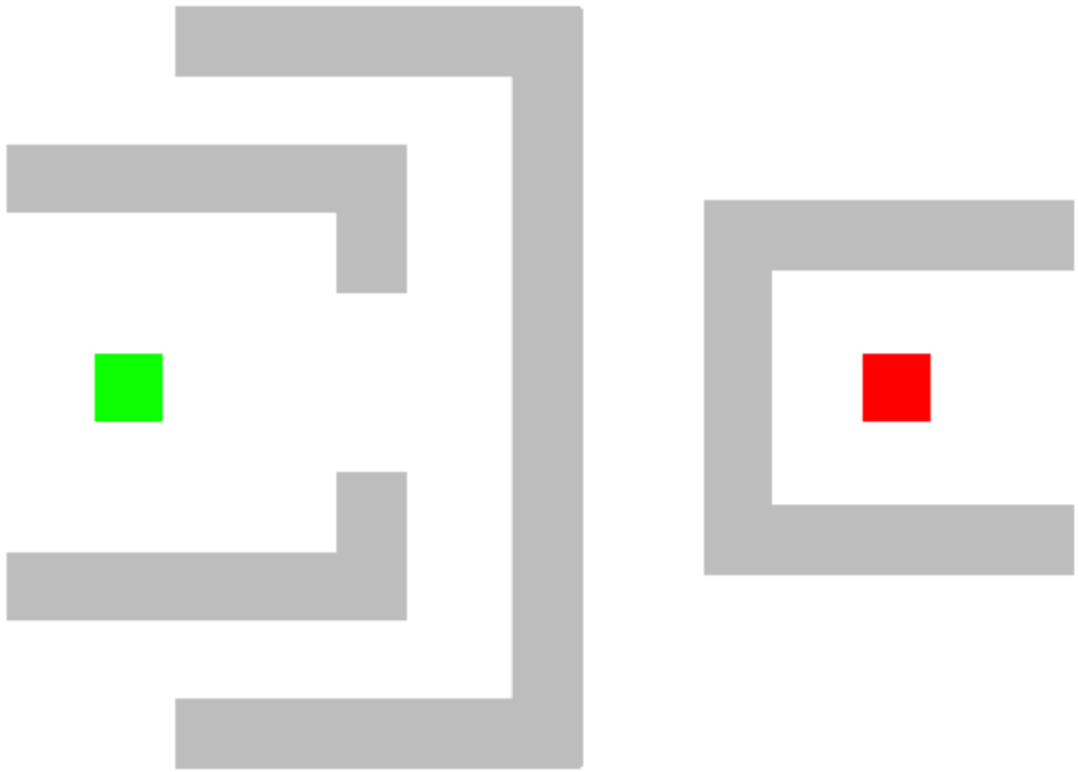
- Path Finding (From 2D to 3D)
- Navigation Meshes
  - Introduction
  - Geometry
  - Search Graph
- NavMesh Path-finding
- Movement Approaches
- Geometry Automation
- Discussion
- Conclusion

Pathfinding

> “A path is a list of instructions for getting from one location to another”

Jeff Huang, Computer Science Department at Brown

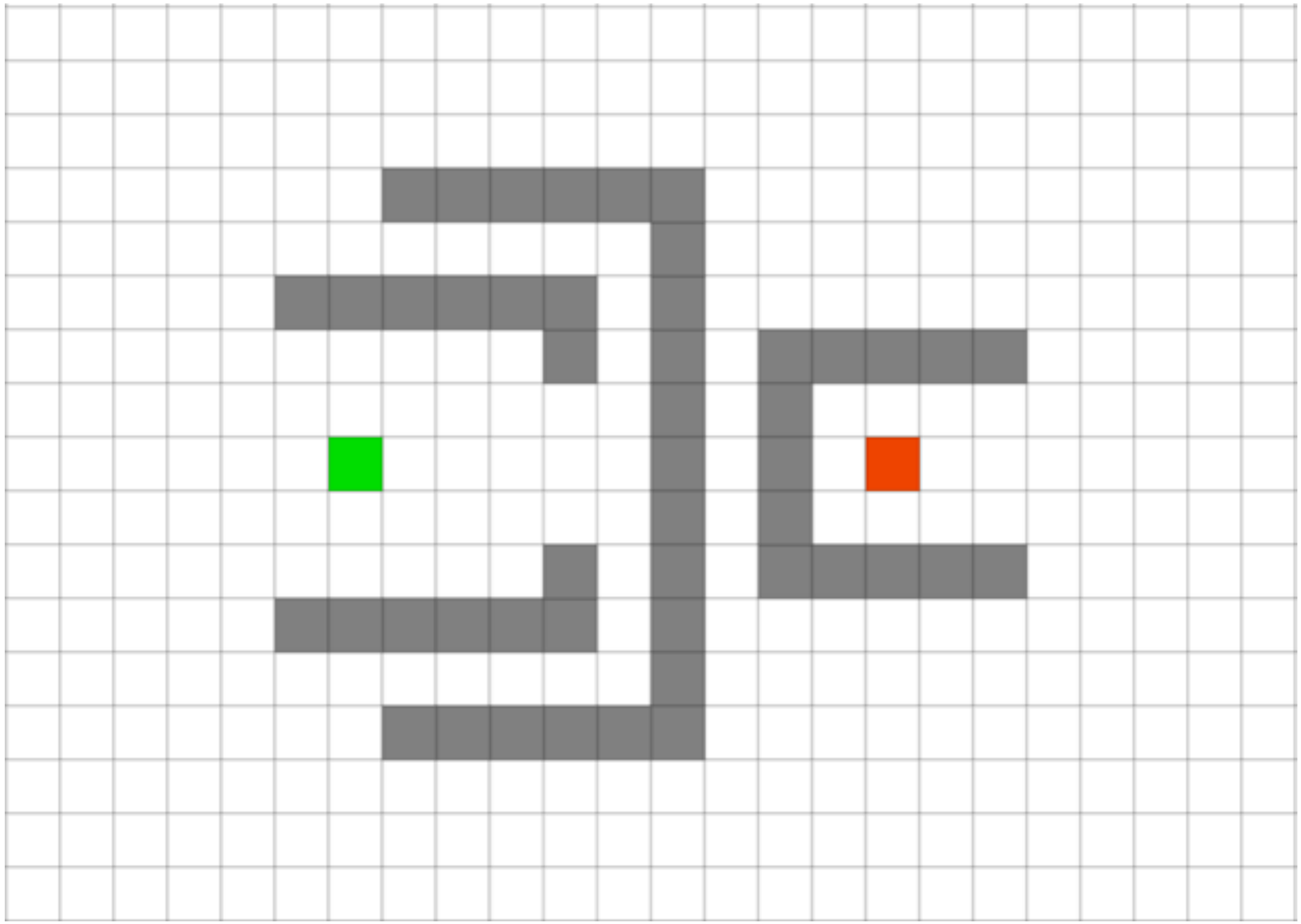
Pathfinding | 2D



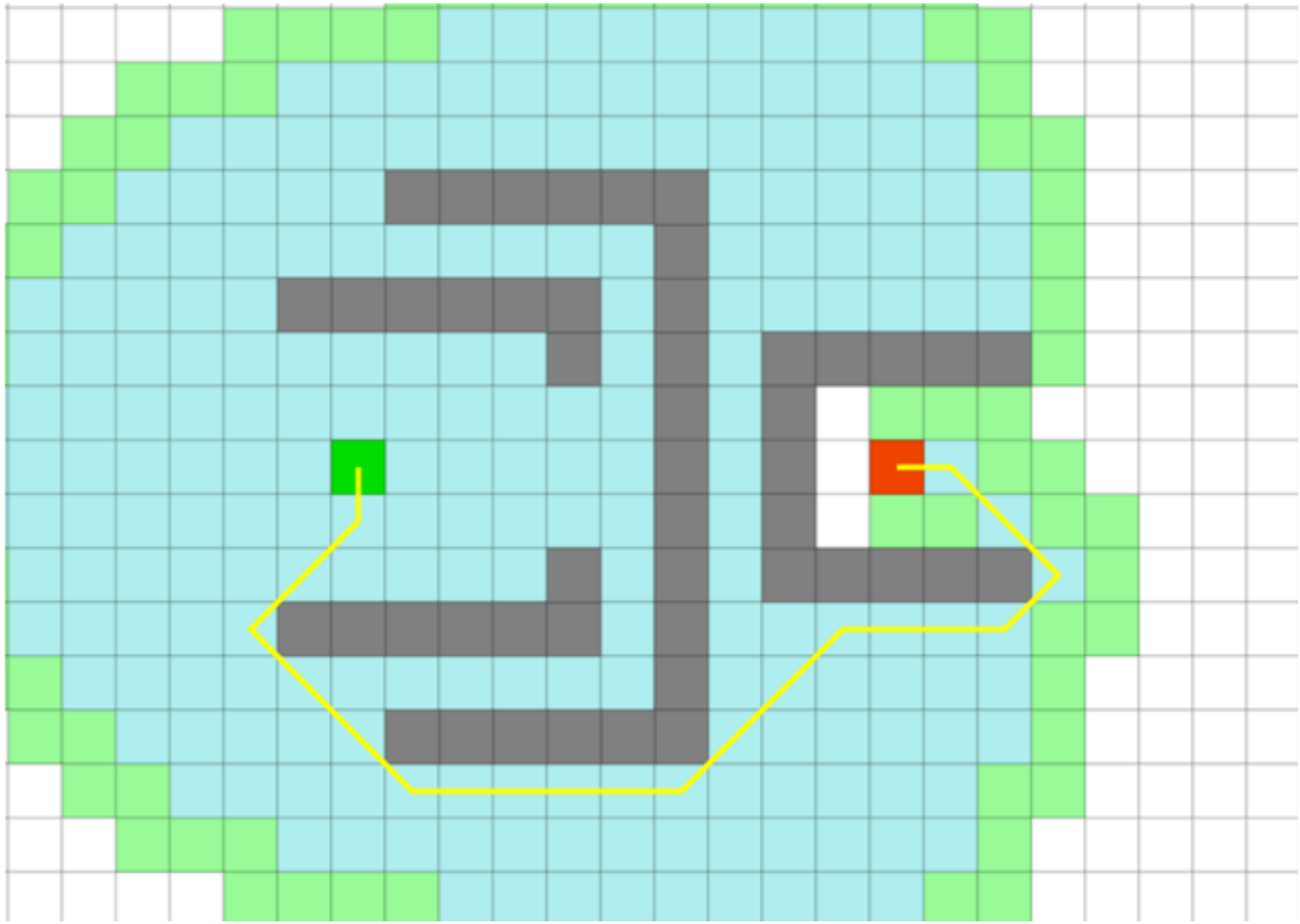
Moving from point A to point B

Legend
- Start
- Target
- Obstacle

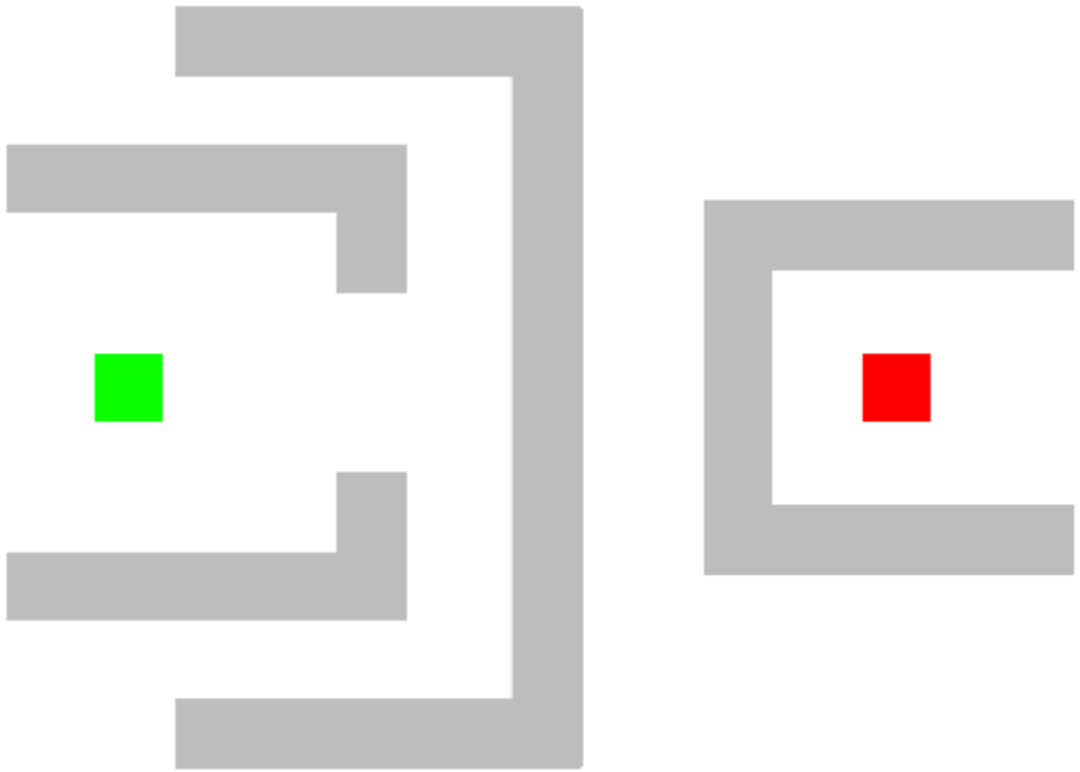
Pathfinding | 2D Grid



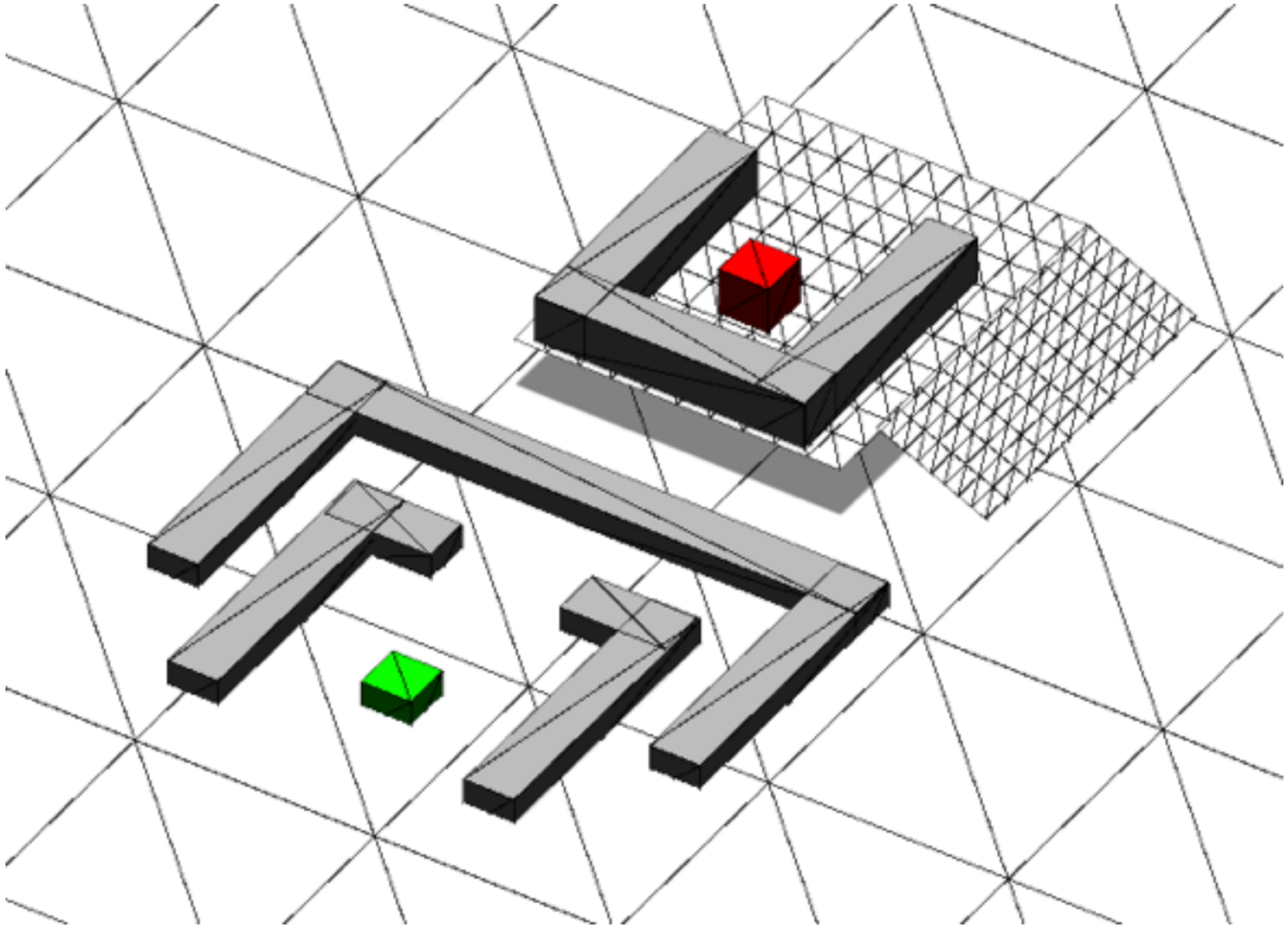
Pathfinding | A Example*



Pathfinding | And what if 3D?



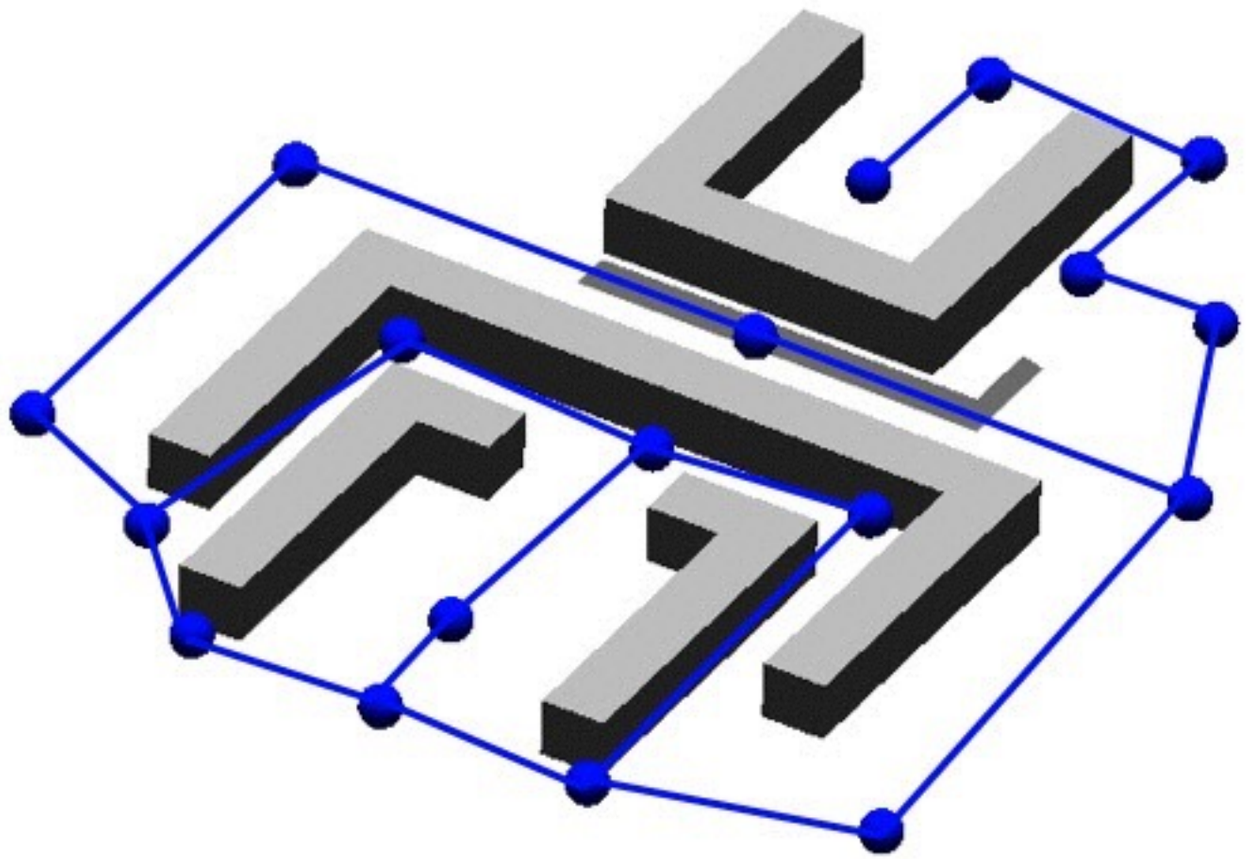
Pathfinding | 3D Mesh Representation



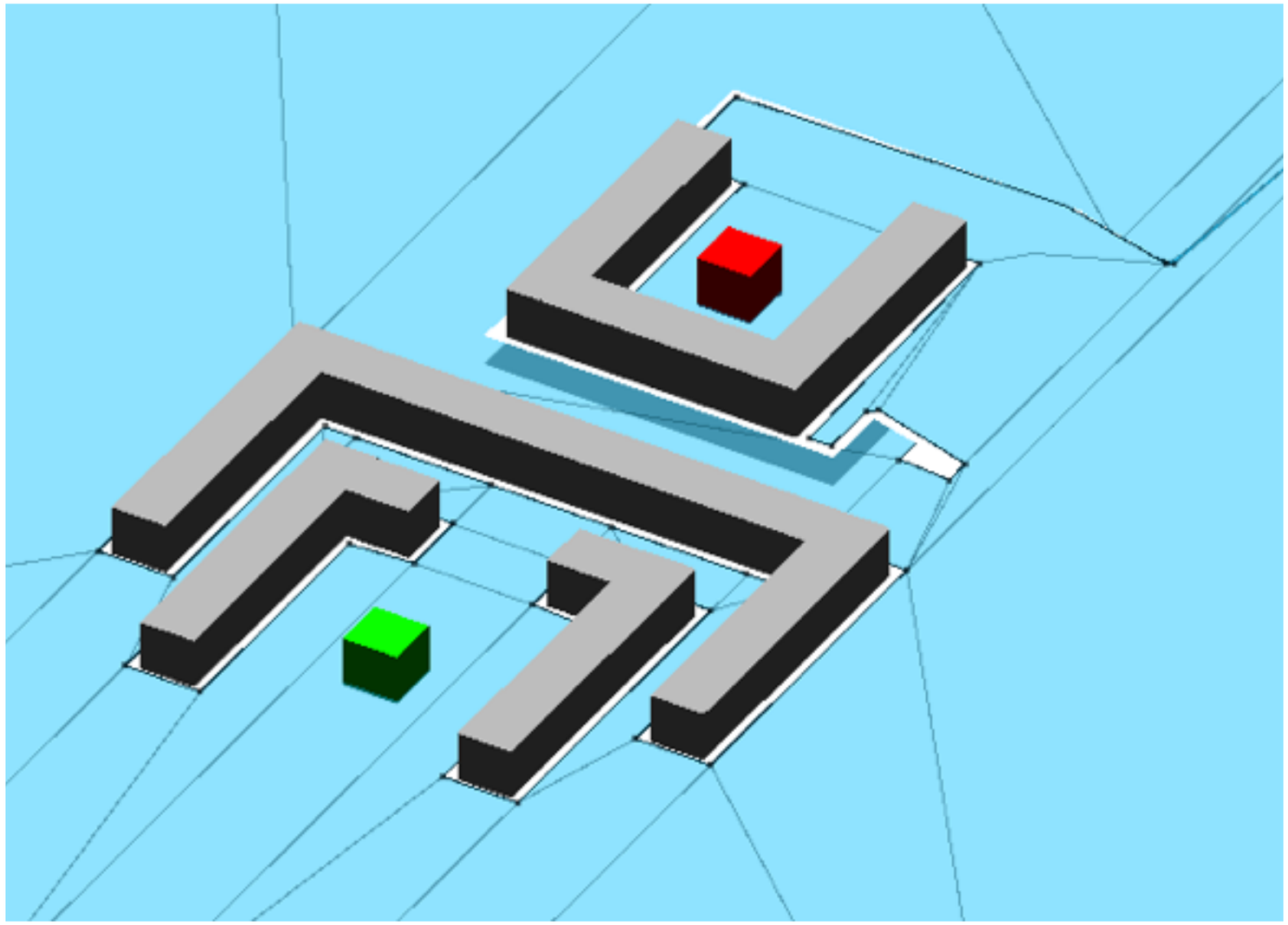
How to apply pathfinding on it?

Legend
- Start
- Target
- Obstacle

Pathfinding | Waypoints



Pathfinding | NavMesh Geometry



What do you see here?

Legend

- Start
- Target
- Obstacle

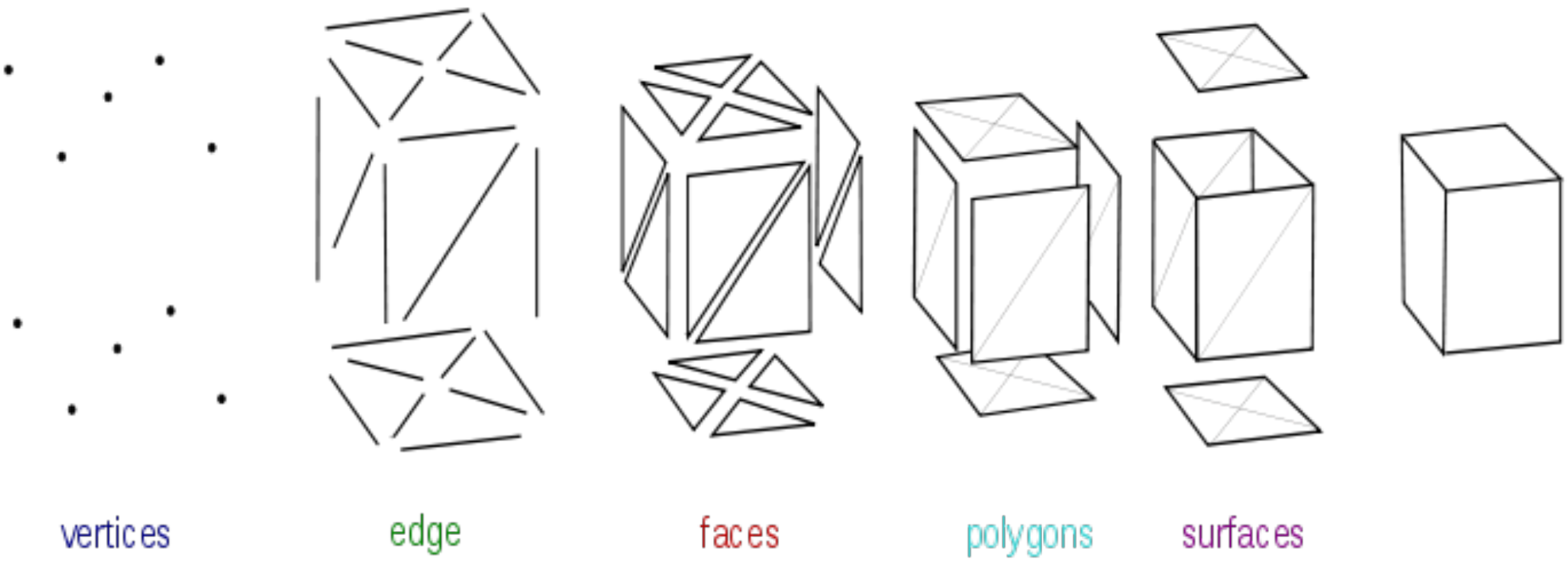
Pathfinding

> “The core pathfinding algorithm is only a small piece of the puzzle, and it’s actually not the most important.”
>
> Paul Tozour, AI Game Programming Wisdom

NavMeshes | *Introduction*

- Convex polygons that describe the “walkable” surface
- Can encode relevant information
- Static Geometry
- Optimise the underlying search space

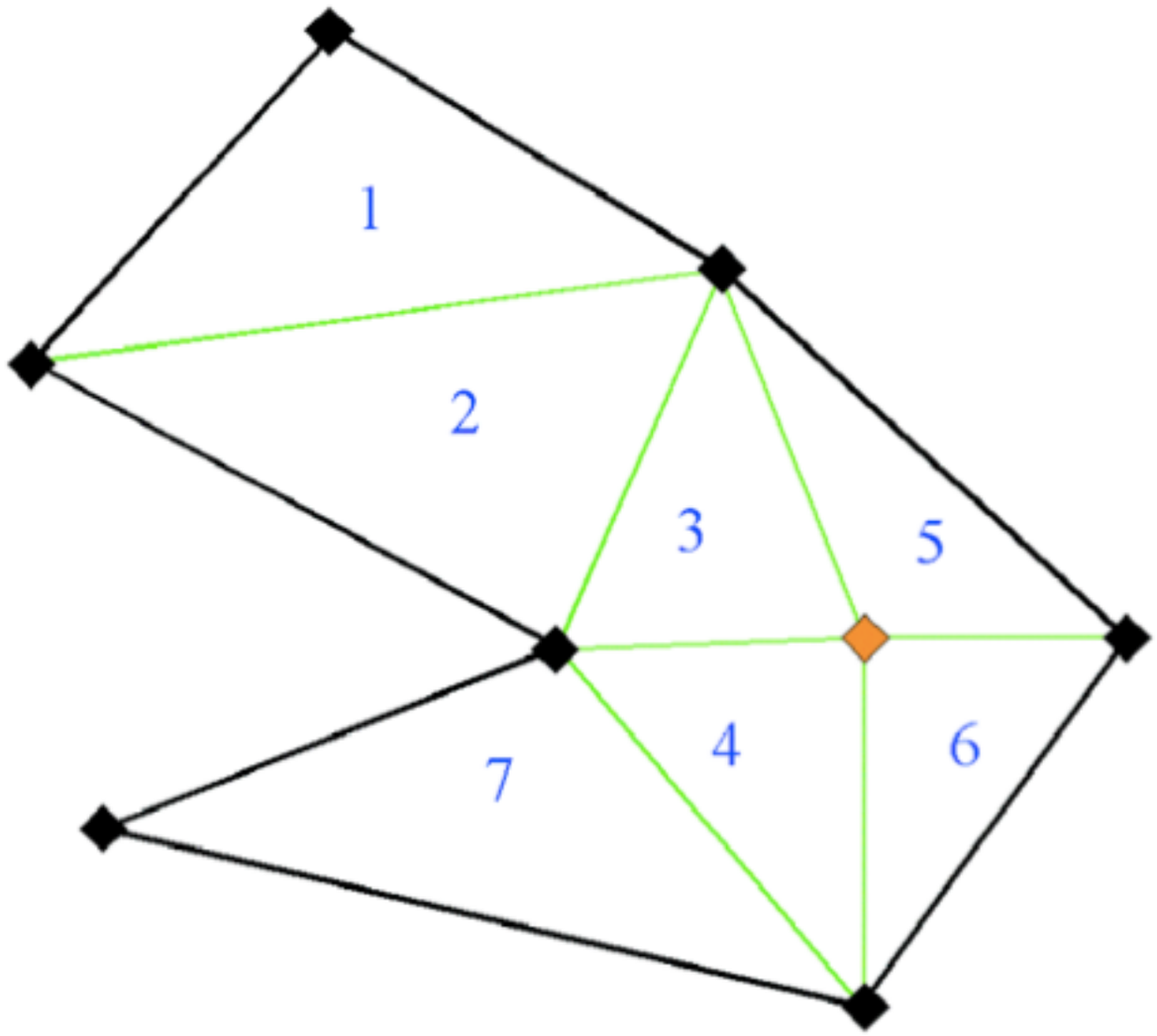
NavMeshes | *Understanding the geometry*



http://en.wikipedia.org/wiki/Polygon_mesh#mediaviewer/File:Mesh_overview.svg

NavMeshes | *Geometric Notations*

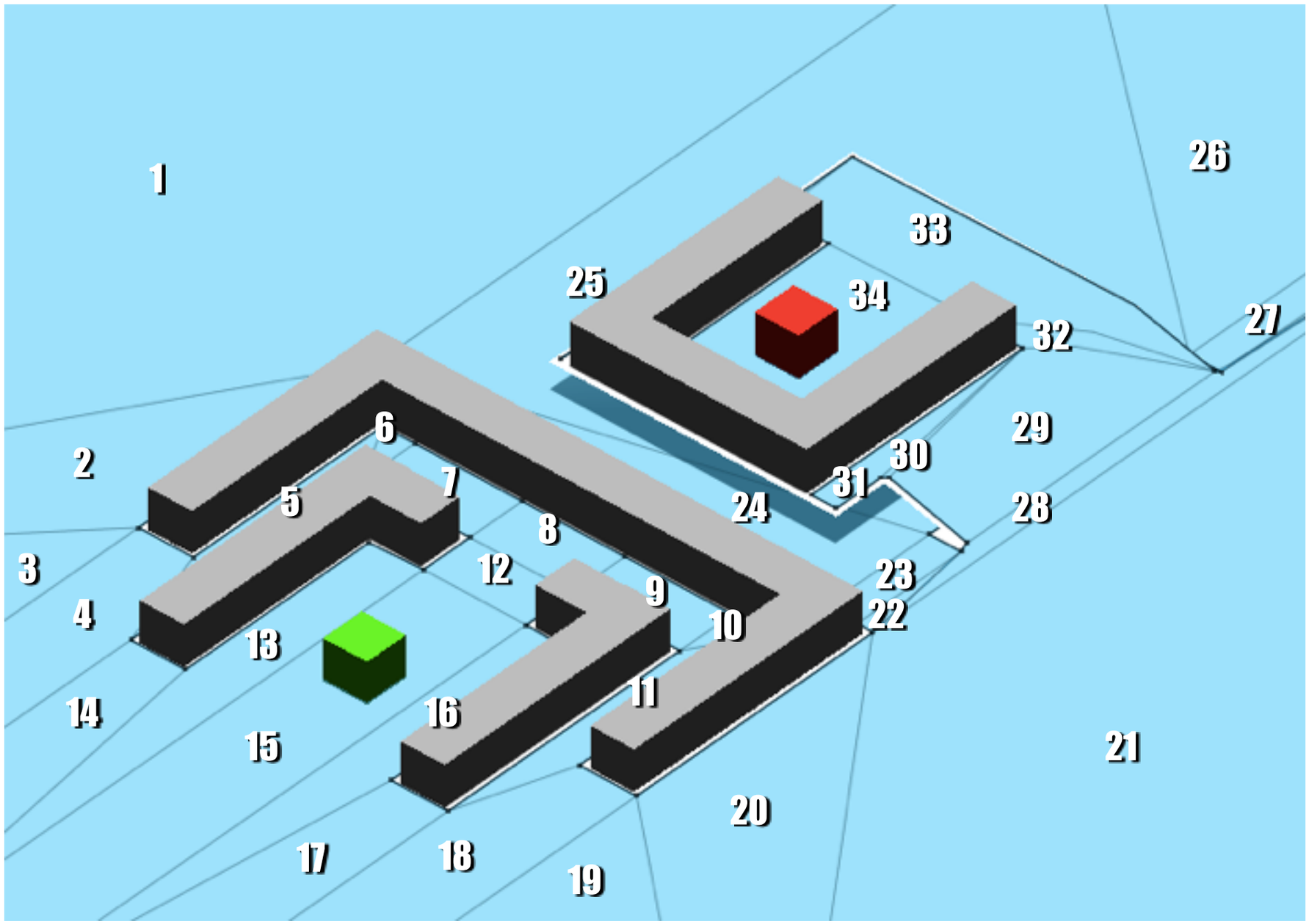
- **Non Solid Edges**
- **Non Solid Vertices**
- **NavTri Edges**



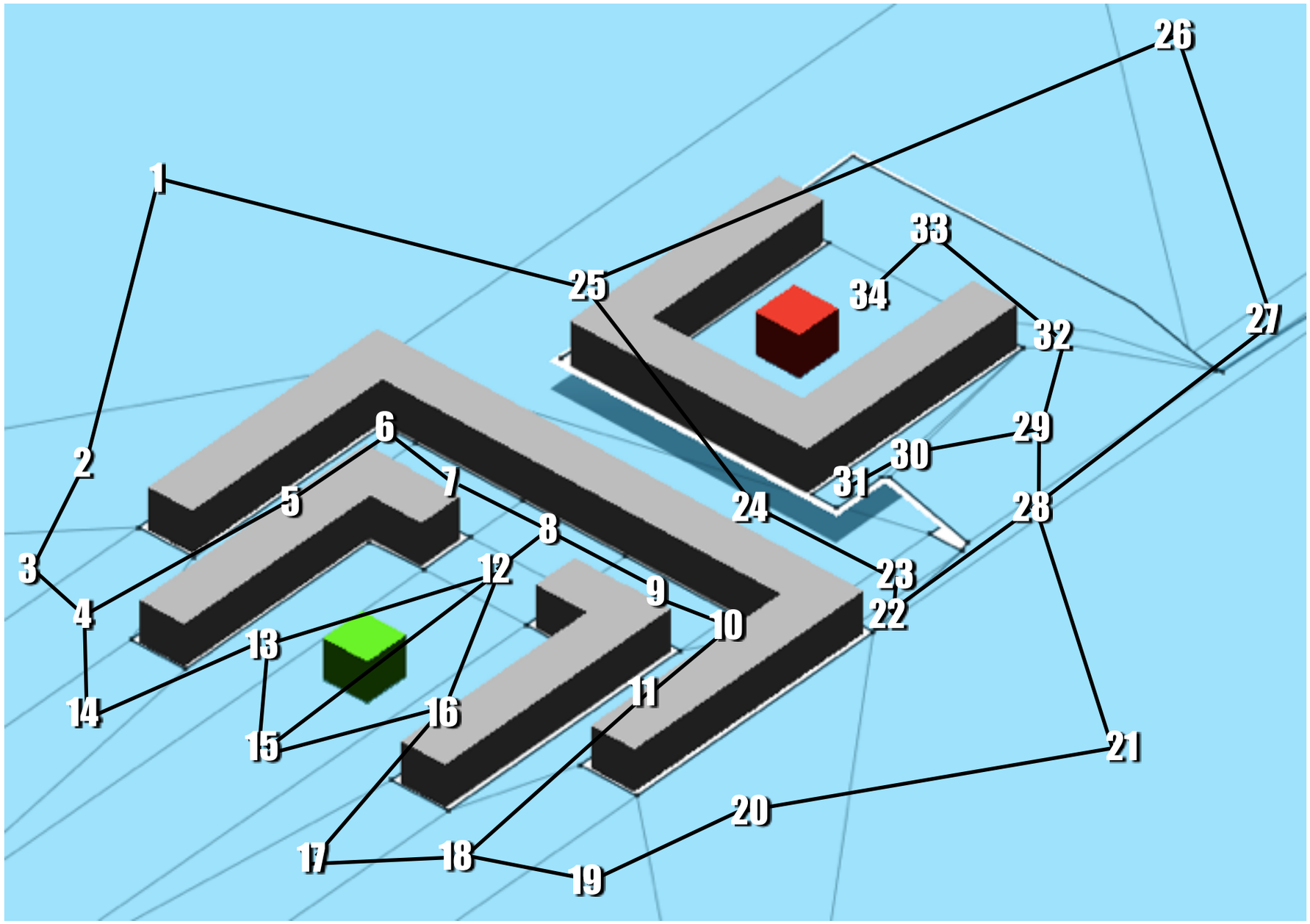
NavMeshes | *Search Graph*

- Virtual world is represented as a graph
- Transcribes the 3D Space representation to a Graph
- Nav Mesh Nodes
- Enables us to use well knows 2D pathfinding algorithms

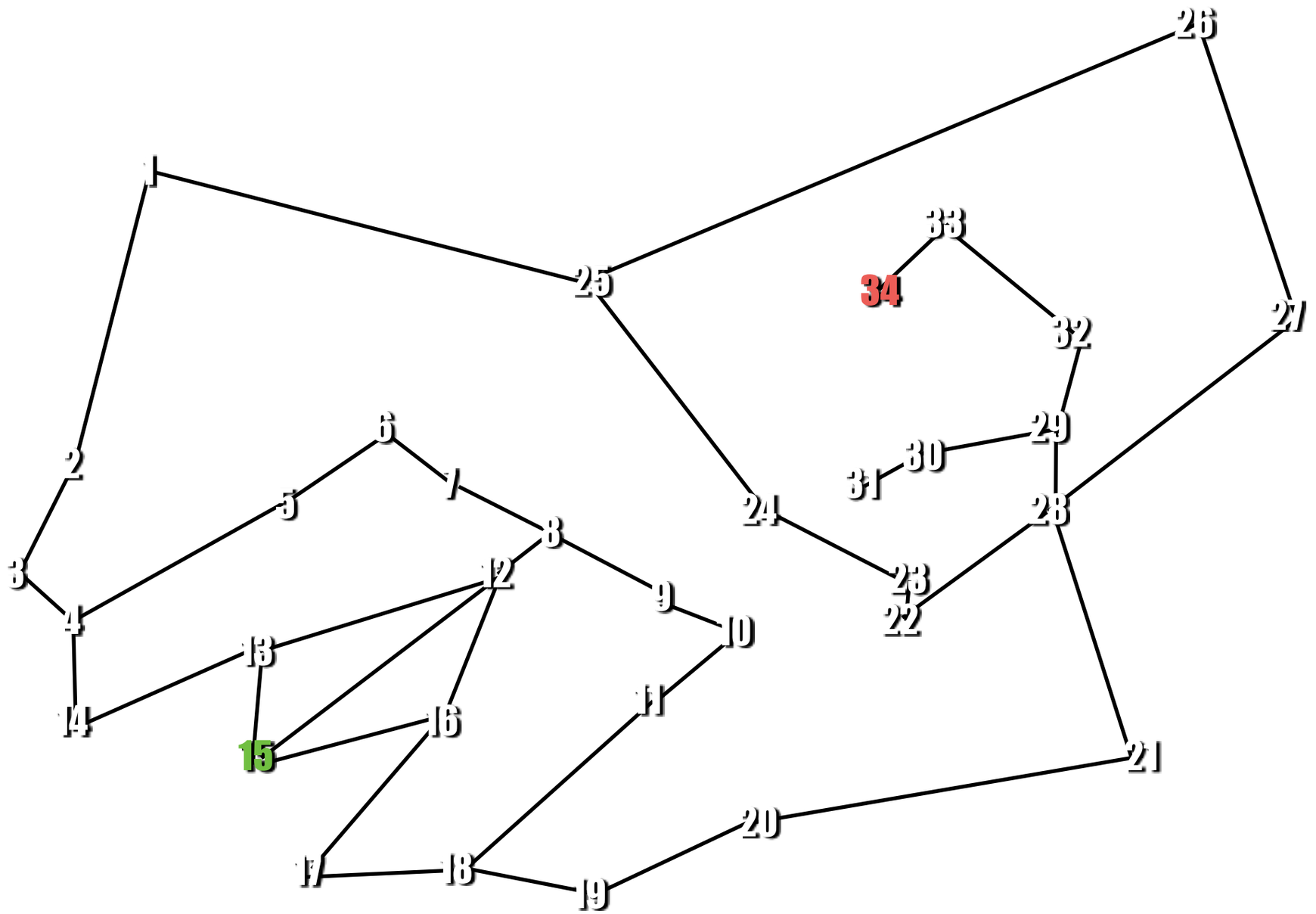
NavMeshes | Graph Example



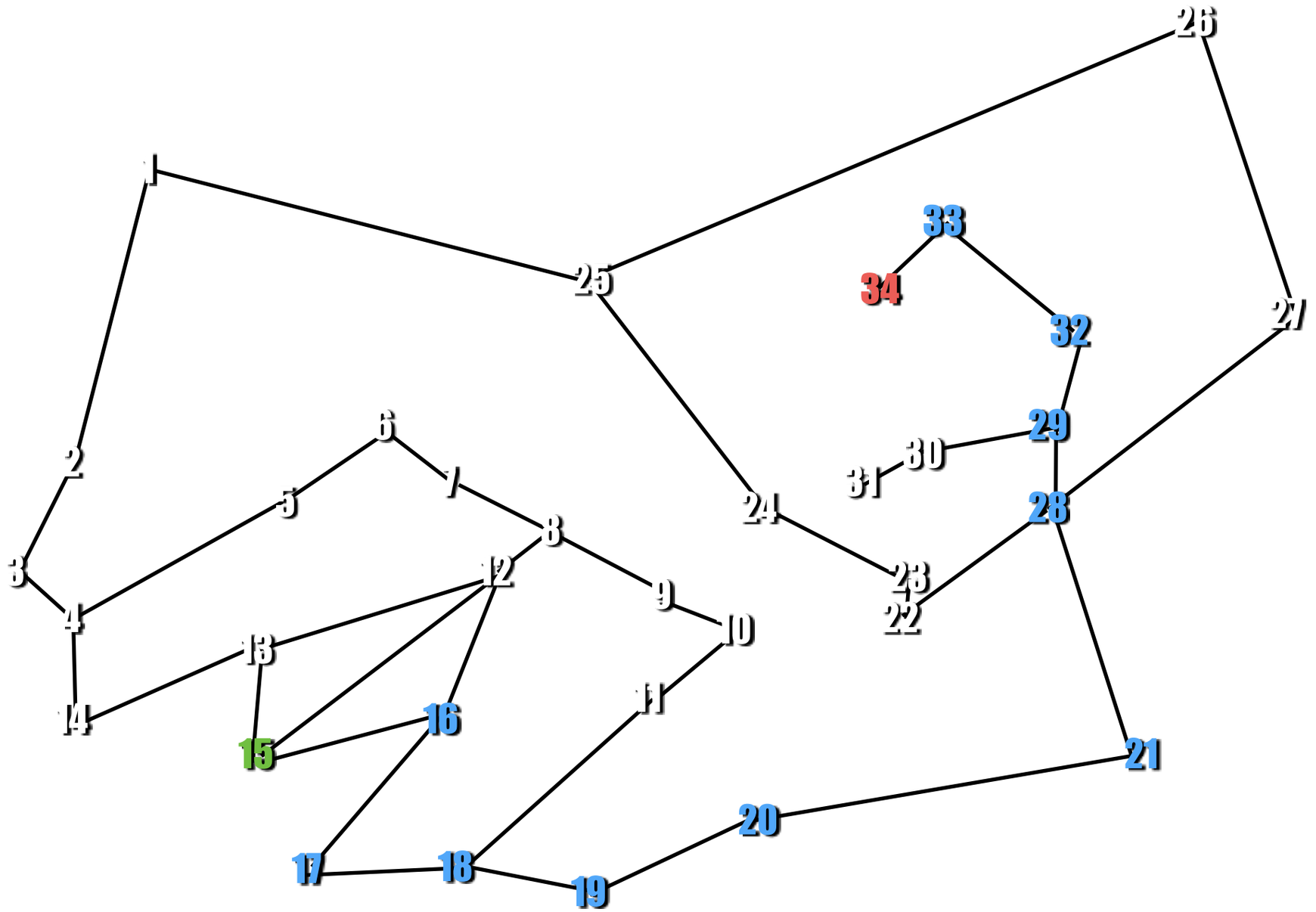
NavMeshes | Graph Example



NavMeshes | Path Finding in a Graph

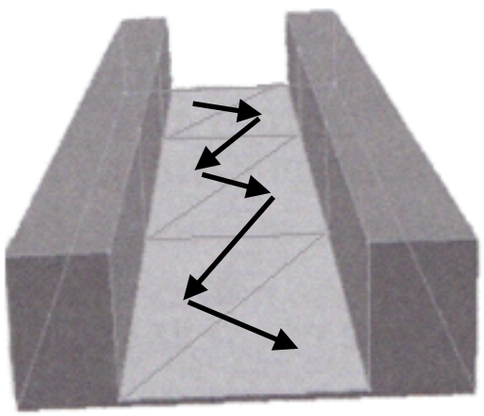


NavMeshes | A*

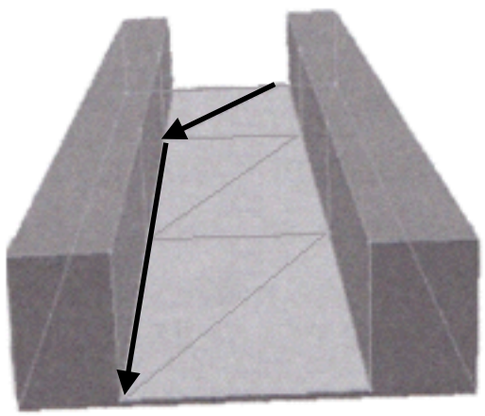

26
1
25
33
34
32
27
6
2
5
7
8
29
30
31
24
28
3
12
9
23
4
22
10
13
11
14
16
15
21
20
17
18
19

NavMeshes | *Movement*

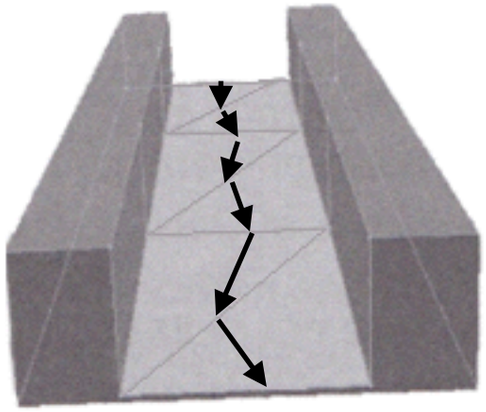
- How to travel between node? (A.K.A. where to step?!)



Poly Center

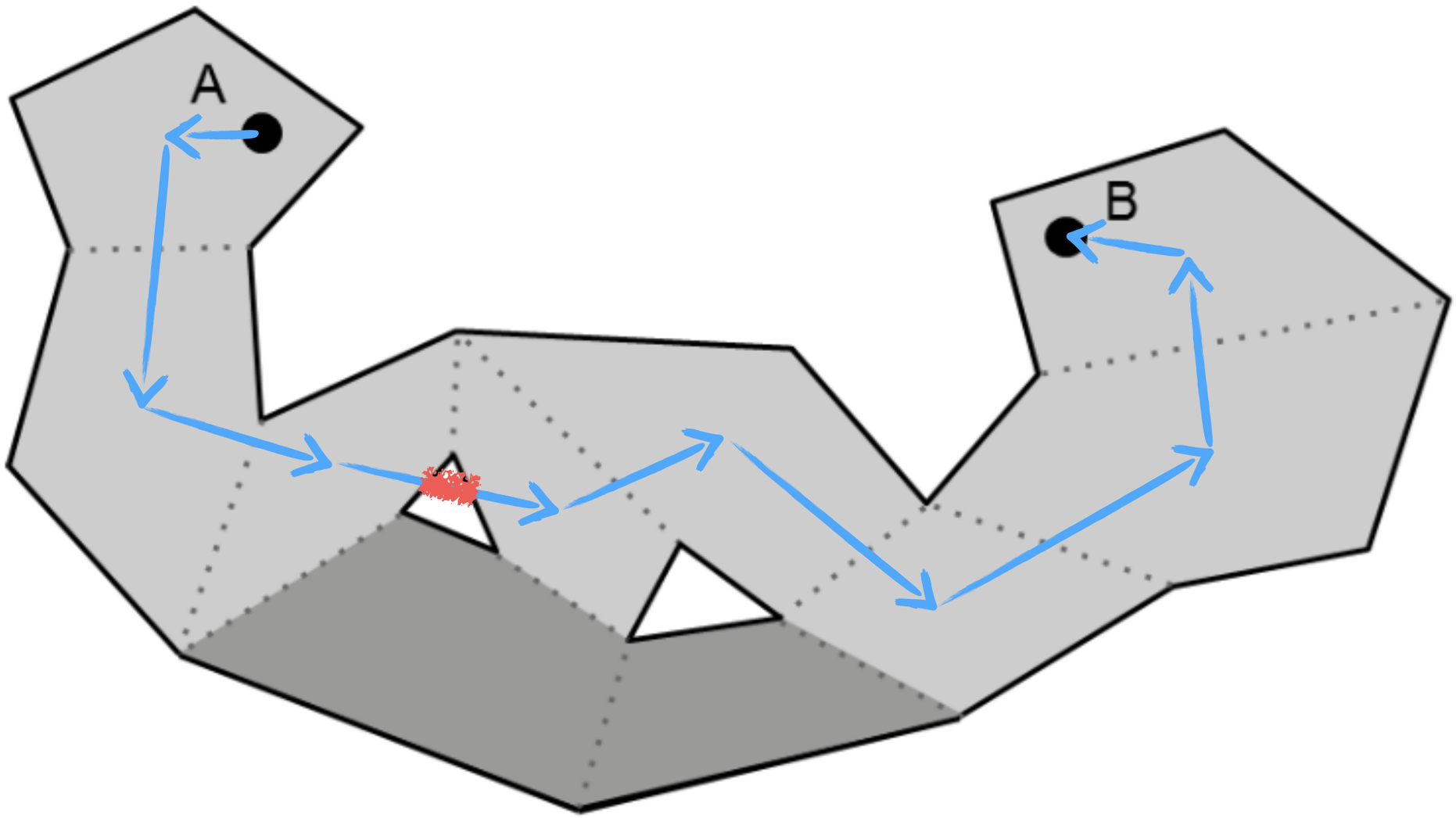


Vertices

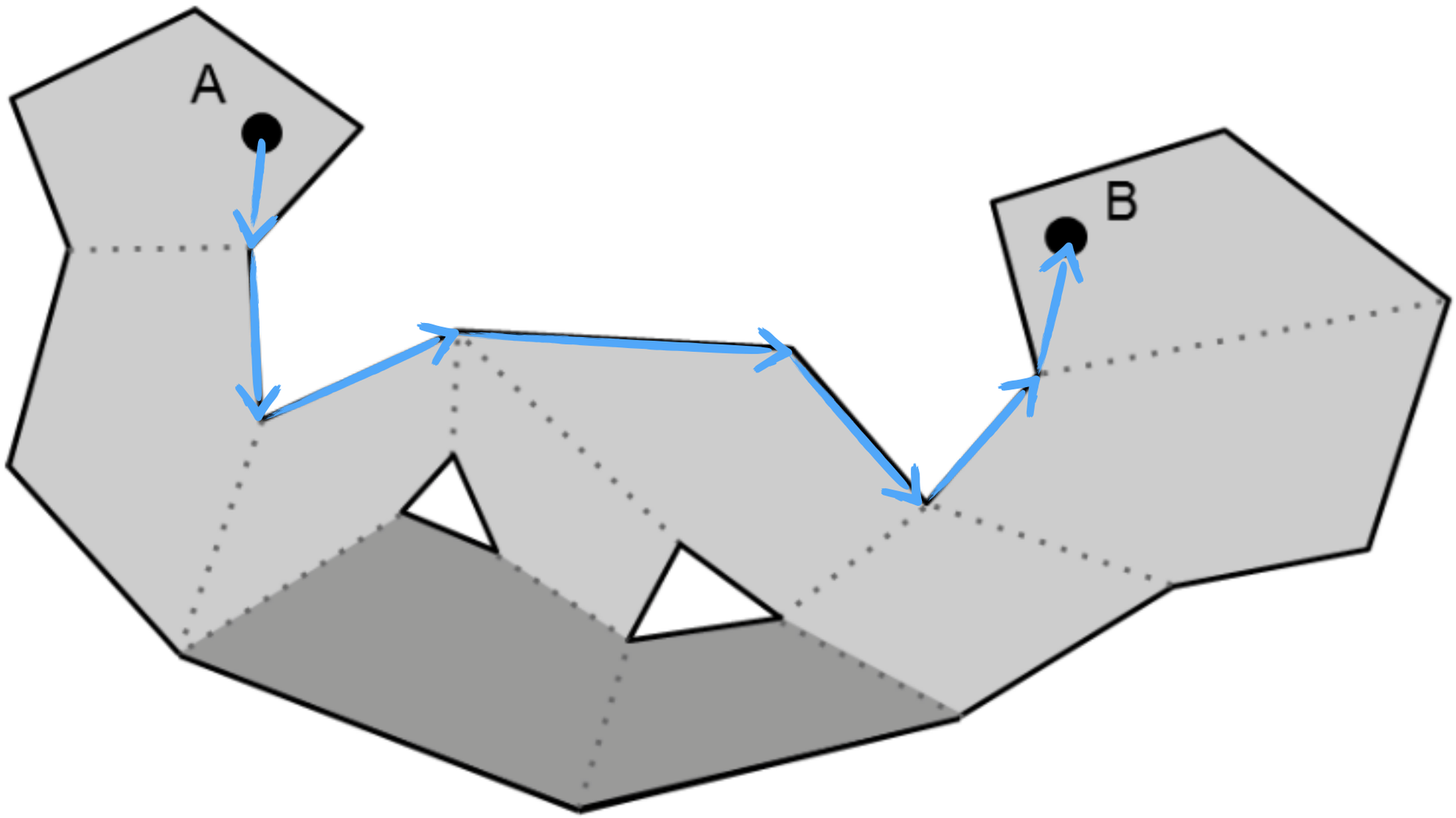


Edges

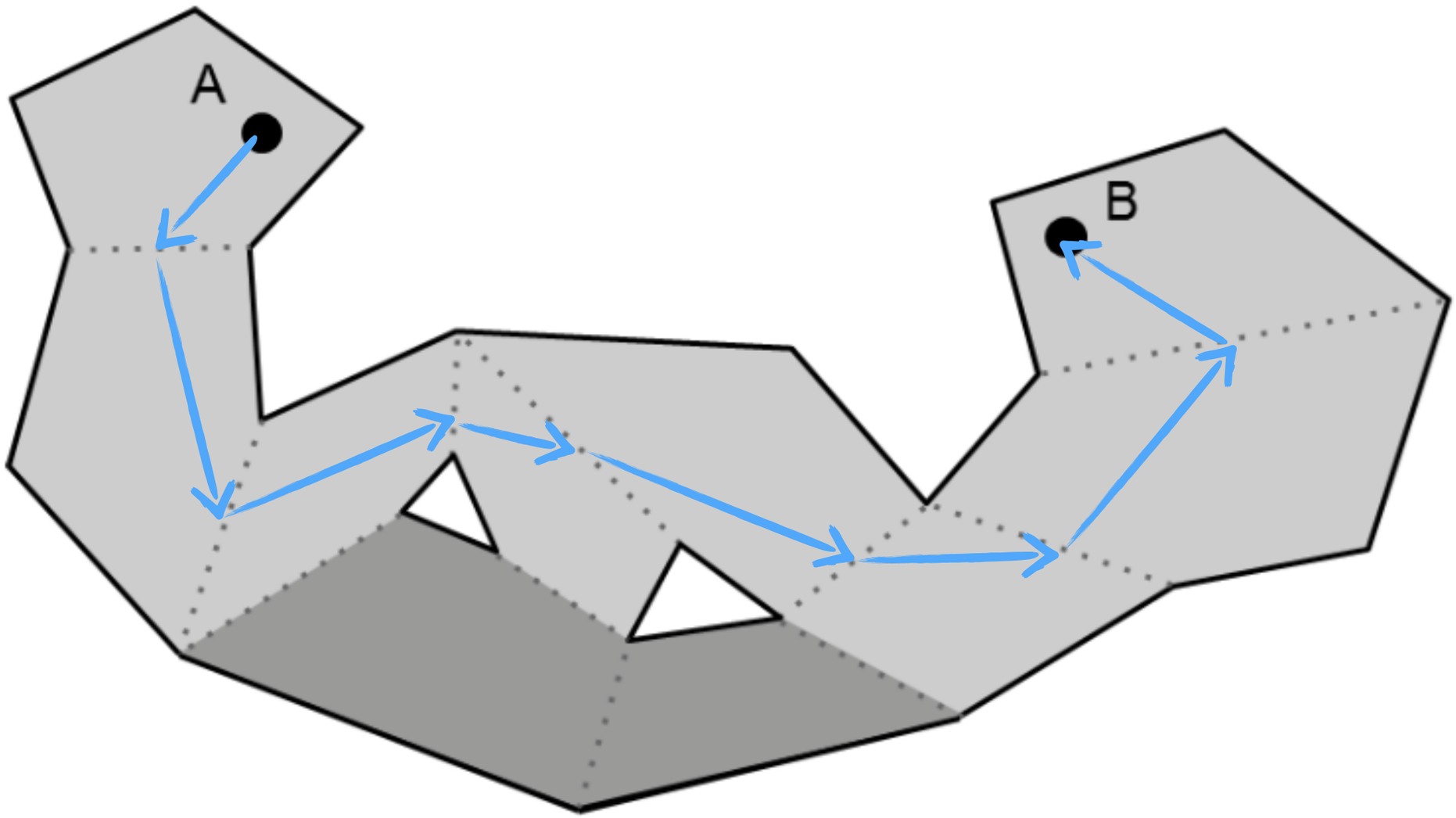
NavMeshes | Poly Center



NavMeshes | Vertices



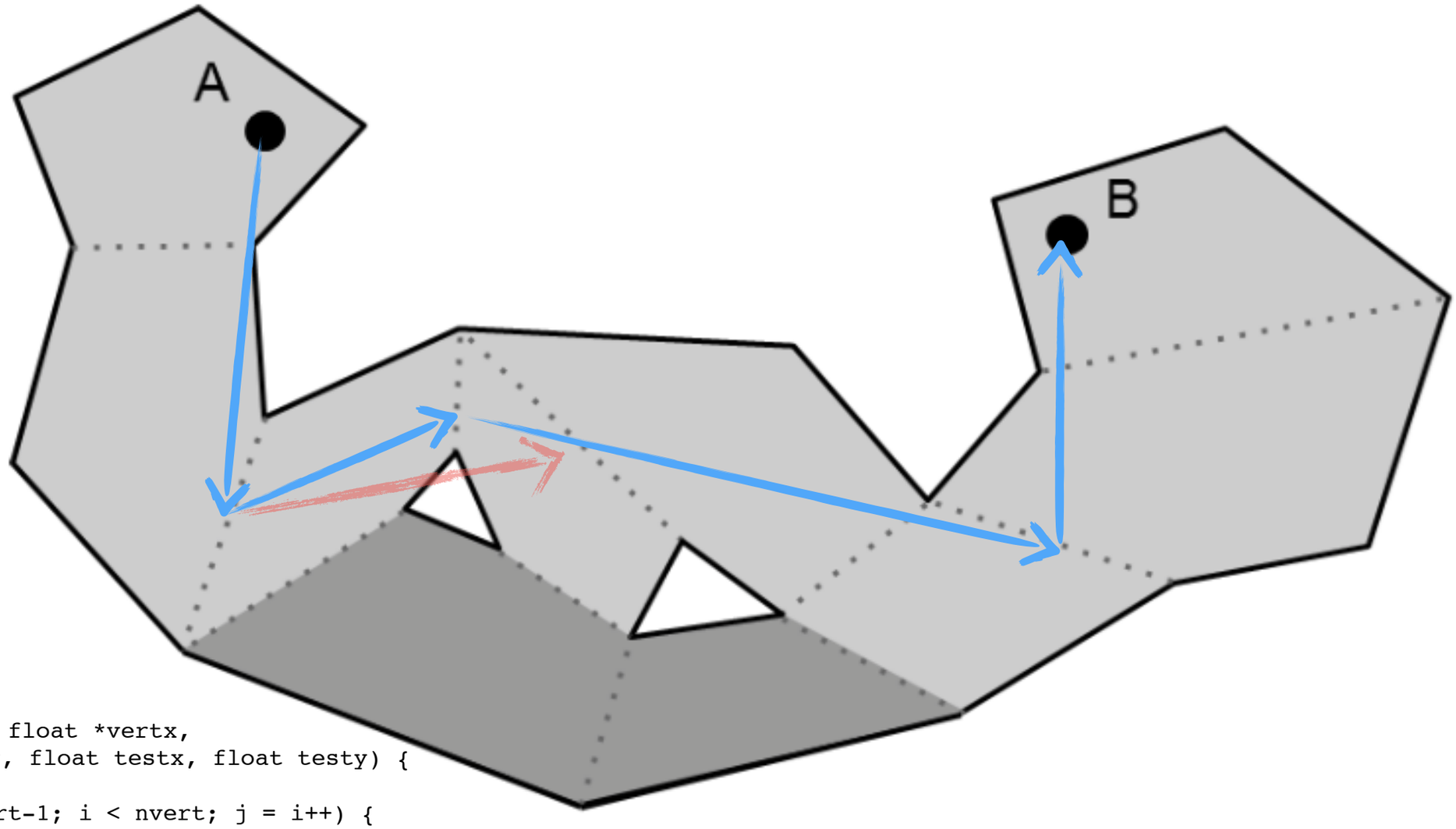
NavMeshes | Edges



NavMeshes | Movement Smoothing

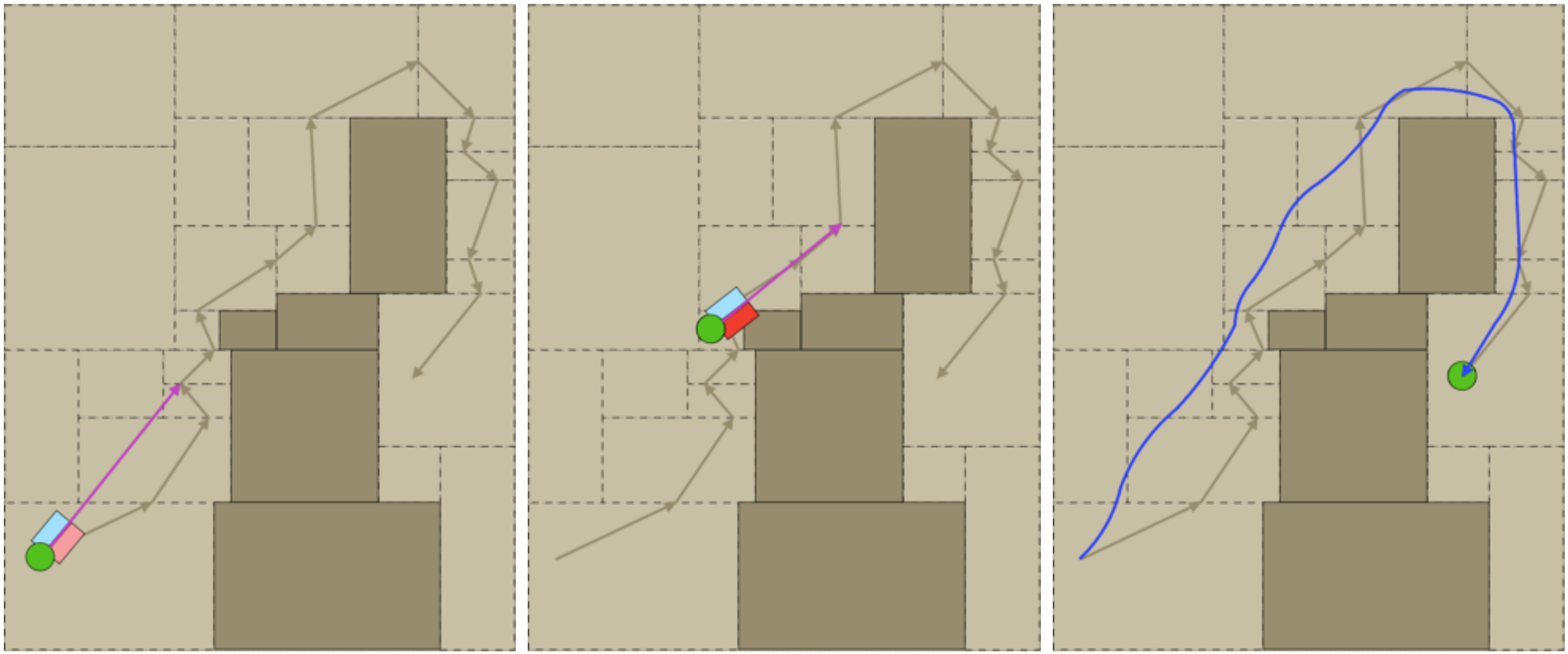
We do have a Path described as 3D “waypoints” but is it enough to provide a believable movement?

NavMeshes | Line of Sight




```
int pnpoly(int nvert, float *vertx,
           float *verty, float testx, float testy) {
  int i, j, c = 0;
  for (i = 0, j = nvert-1; i < nvert; j = i++) {
    if ( ((verty[i]>testy) != (verty[j]>testy)) &&
        (testx < (vertx[j]-vertx[i]) * (testy-verty[i]) / (verty[j]-verty[i]) + vertx[i]) )
       c = !c;
  }
  return c;
  //http://www.ecse.rpi.edu/Homepages/wrf/Research/Short_Notes/pnpoly.html
```

NavMeshes | Reactive Path Following

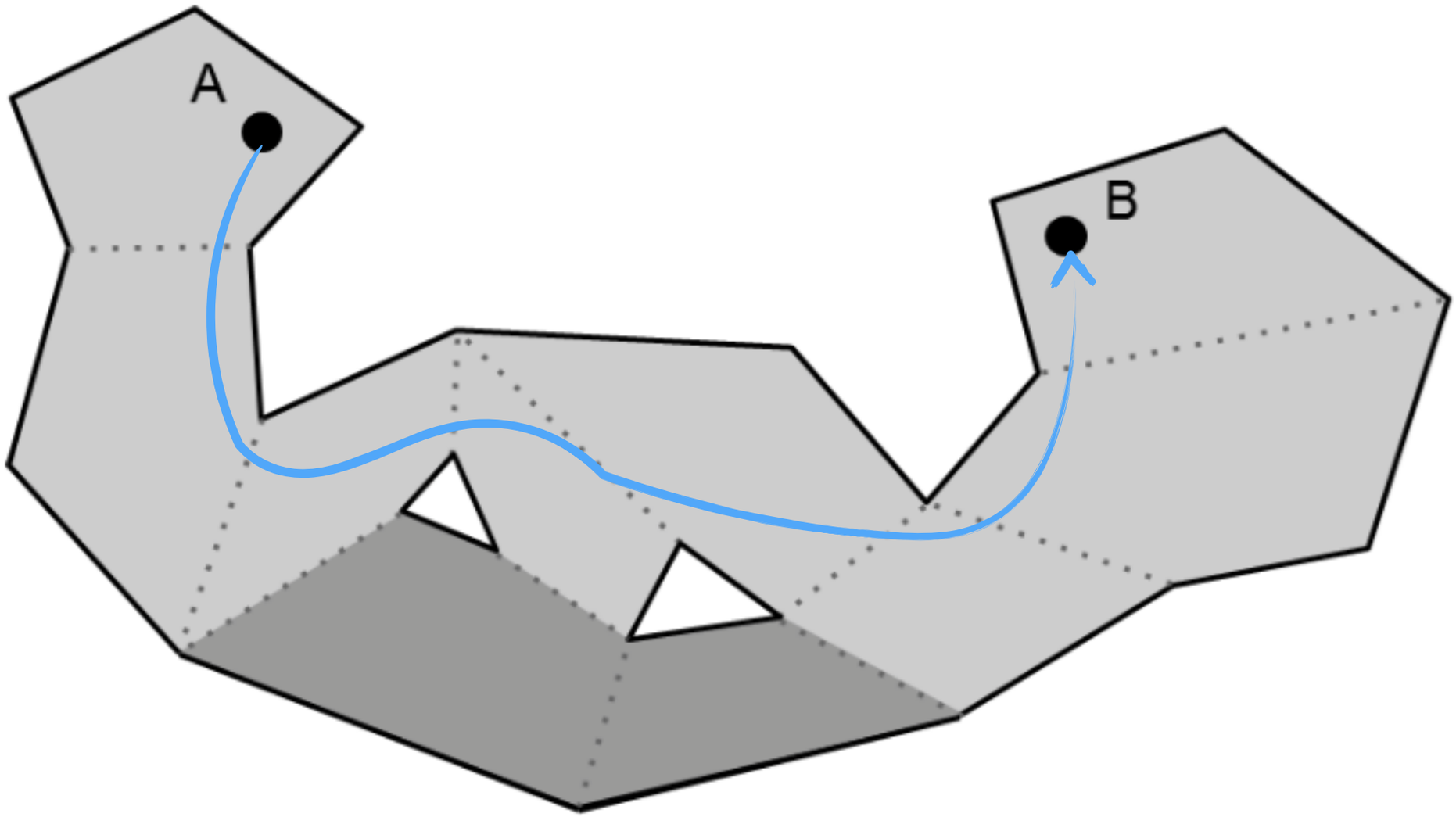


Michael Booth, Valve

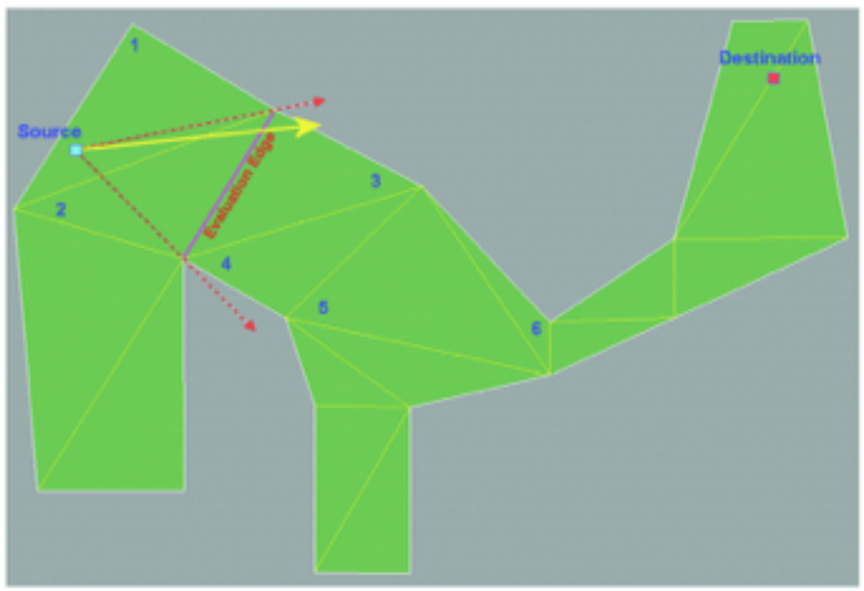
NavMeshes | Reactive Path Following

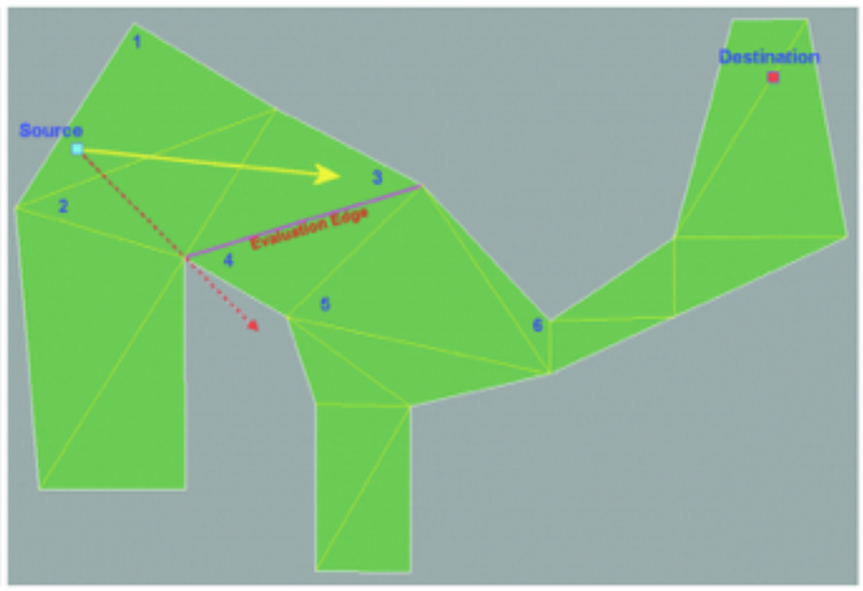
```
// Rough example of the path update function.
// From https://gamingme.wordpress.com/tag/reactive-path-following/
void Bot::PathUpdate(f32 m_fDeltaTime)
{
    // unless the bot got too far from the path, we don't have to compute it
    // again. In this case, the realism of the algorithm helps performance!
    // Of course we also have to find a path if we don't have one yet.
    if (!HasPath() || (DistanceFromPath() > m_fMaxPathDistance))
    {
        m_lPathNodes = AStar::FindPath(
            m_pWorld->GetNavigationMesh(),
            m_vPosition,
            m_vTarget);
        m_iCurrentNode = 0;
    }

    // looking for the best node to track. Notice that the next node is local,
    // the decision is made every frame.
    Node* pNextNode = m_lPathNodes[m_iCurrentNode];
    for (s32 i = m_iCurrentNode; i < m_lPathNodes.GetSize(); ++i)
    {
        if (IsFacing(m_lPathNodes[i]) && IsClearPath(m_lPathNodes[i]))
        {
            pNextNode = m_lPathNodes[i];
        }
    }

    // assuming some complex heuristic might take place to decide which
    // priority each system gets, we don't update the member variables
    // directly. Instead, make requests informing the current system
    RequestMovement(
        ComputePathTranslation(m_fDeltaTime),
        ComputePathSteering(pNextNode, m_fDeltaTime),
        PATH_FOLLOWING);
}
```

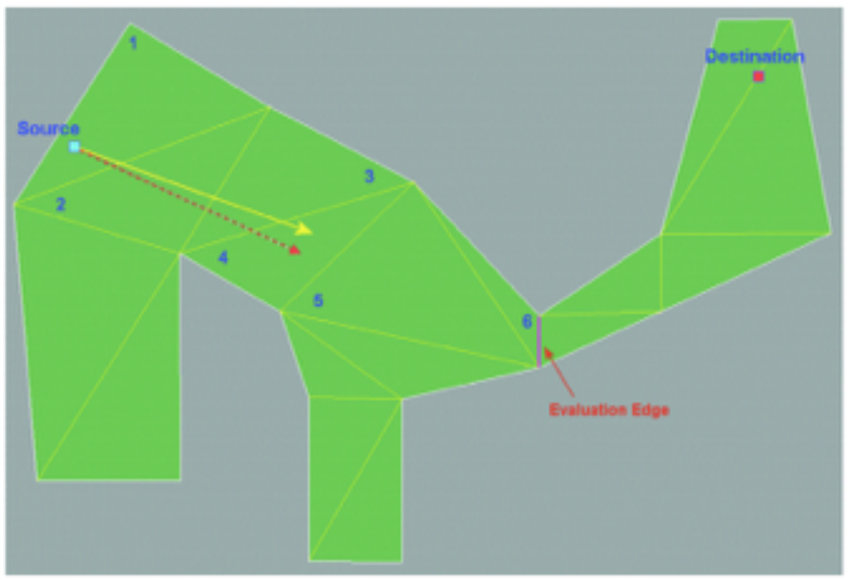
NavMeshes | Reactive Path Following



NavMeshes | Iterative Constraint Rays



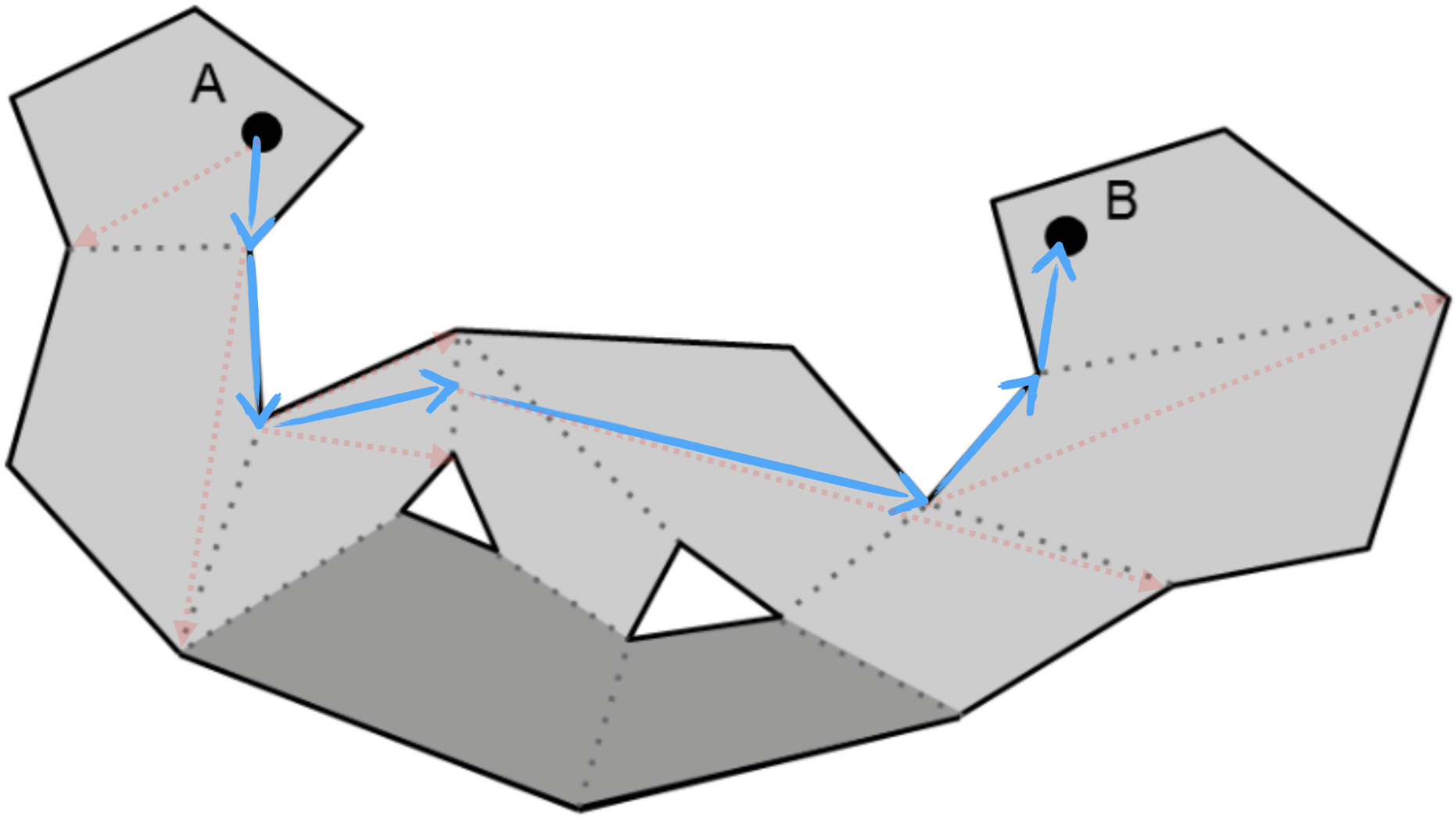




1. Set edgeCurrent equal to the path solution from the current triangle to the destination triangle.
2. If edgeCurrent has one or more solid vertices, define the initial containment rays as normalised vectors from the starting point to each of the solid vertices.
3. If only one vertex is solid on edgeCurrent, negate the alternate constraint ray and bias the direction toward edgeCurrent. This prevents cosine evaluations of 180-degree angles.
4. If no vertices are solid on the current edge, the containment rays are left empty and this initialisation step is deferred until the first solid vertices are encountered in subsequent iterations.

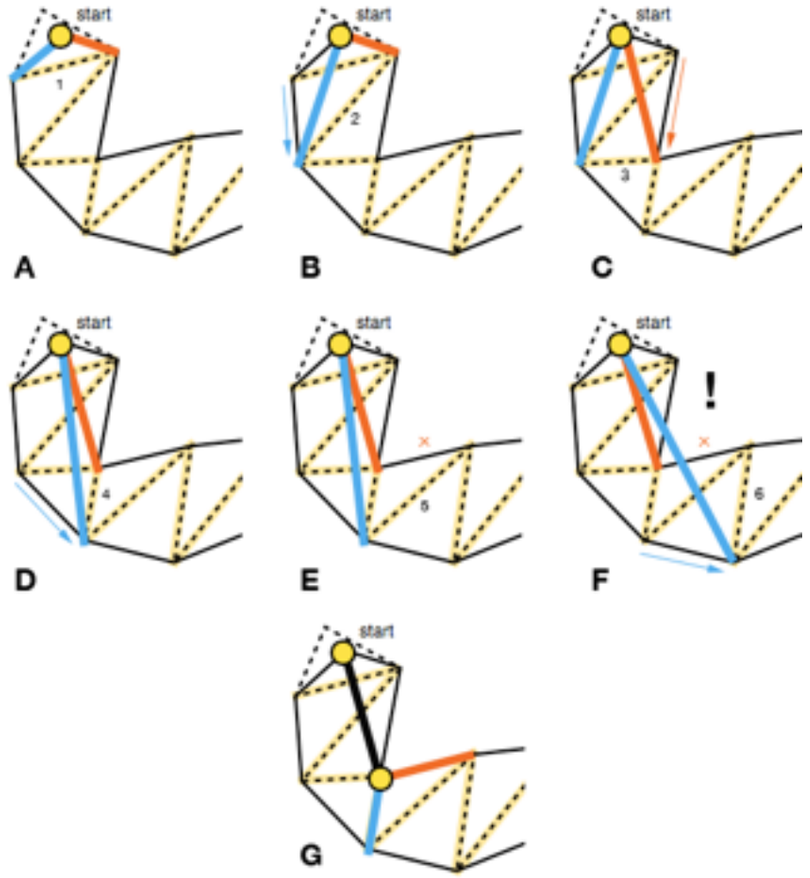
John C. O'Neill

NavMeshes | Iterative Constraint Rays



3 Iterations

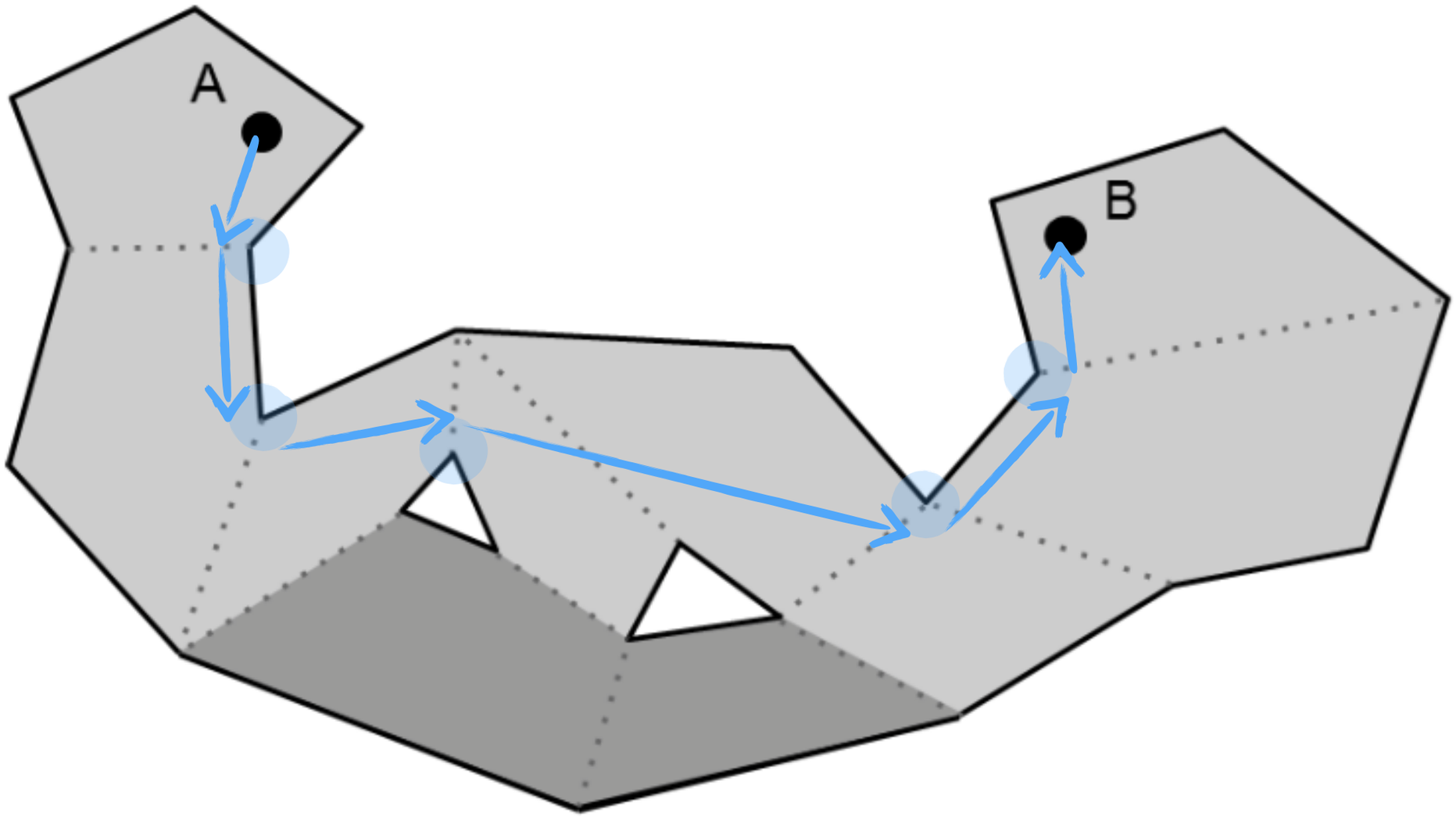
NavMeshes | Simple Stupid Funnel Algorithm



1. Check if the left and right vertices are inside the current funnel;
2. If the new left endpoint is outside the funnel, the funnel is not update (E-F)
3. 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 funnel at the right funnel point location and restart the algorithm from there (G).

Mikko Mononen

NavMeshes | Simple Stupid Funnel Algorithm



NavMeshes | *Automatic Creation*

- Build a NavMesh from virtual world geometry
- Optimal convex partition by Tozour
- Steps:
  - Merging Neighbour Nodes
  - 3 -> 2 Merging
  - Culling Trivial Nods
  - Recursive Subdivision
  - Re-Merging

FIGURE 4.3.2 *Merging polygons into a single convex polygon.*

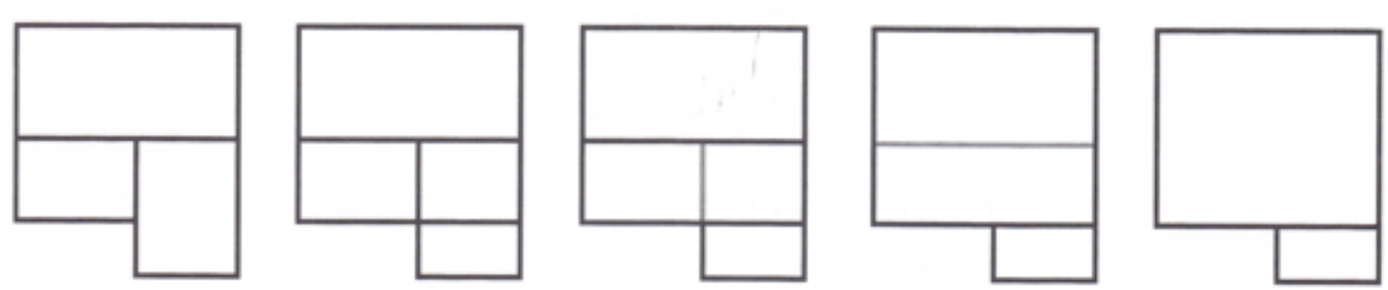


FIGURE 4.3.3 *Converting three convex polygons into two convex polygons.*

NavMeshes | Optimality

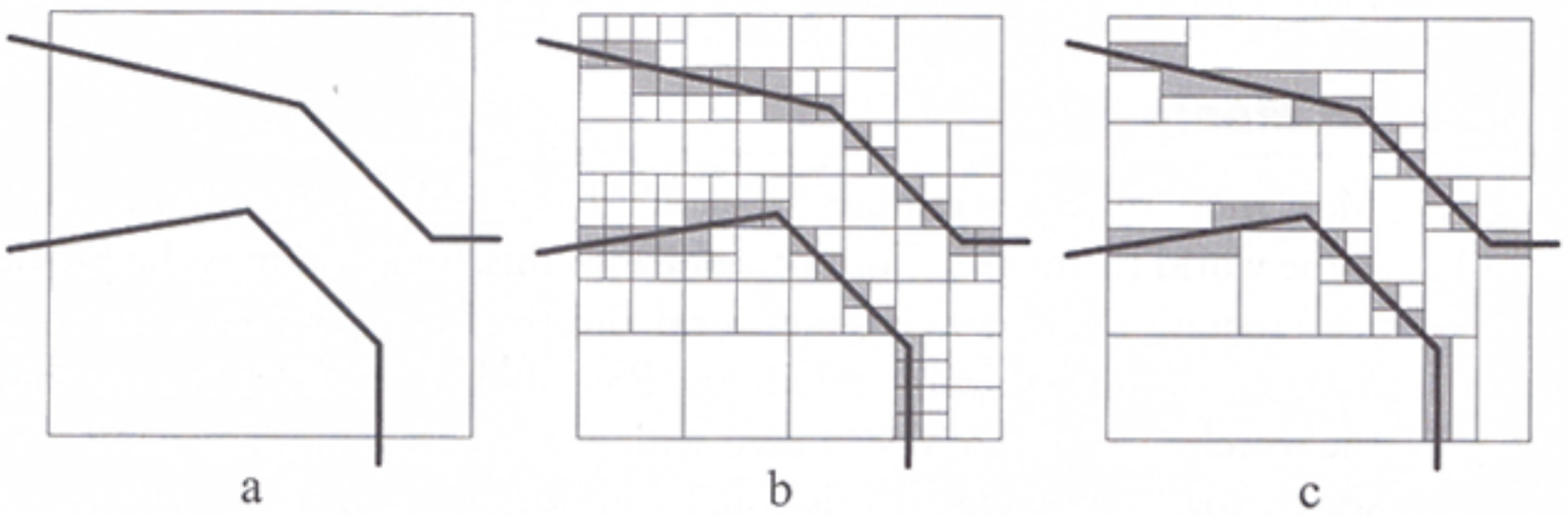


FIGURE 2.2.1 *A floor polygon with walls is subdivided recursively using Tozour's technique, and merged back.*

FIGURE 2.2.2 *An optimal mesh for the example shown in Figure 2.2.1 would contain only six polygons.*

NavMeshes | Polygon Subdivision Algorithm

- By Fredrik Farnstrom (Rockstar)

1. *Partition the geometry into a ground mesh and obstruction mesh (Figure 1)*
2. *Merge polygons within each mesh to reduce the data set*
3. *Fuse together overlapping ground polygons*
4. *Subdivide the ground mesh with its own extrusion and merge*
5. *Extrude the obstruction mesh*
6. *Subdivide the ground mesh with the obstruction mesh and merge (Figure 2)*
7. *Find connections between ground mesh polygons*
8. *Remove parts of the mesh that are disconnected from the rest of the mesh*

NavMeshes | Polygon Subdivision Algorithm

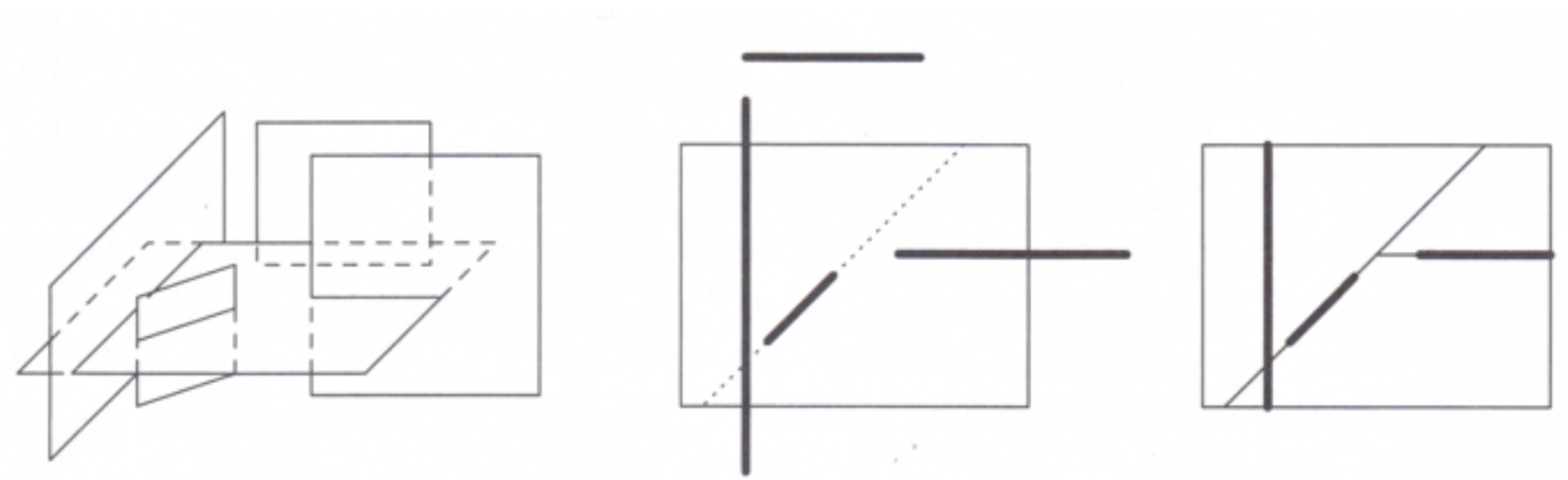


Figure 1

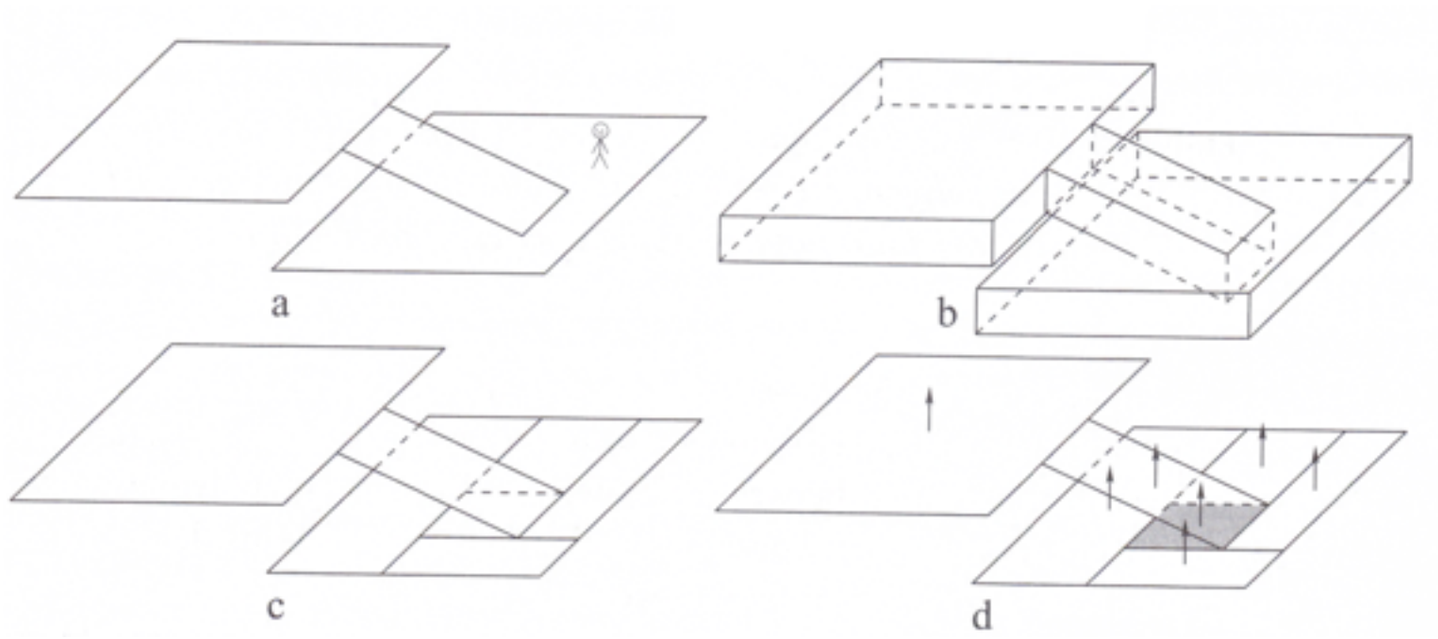


Figure 2

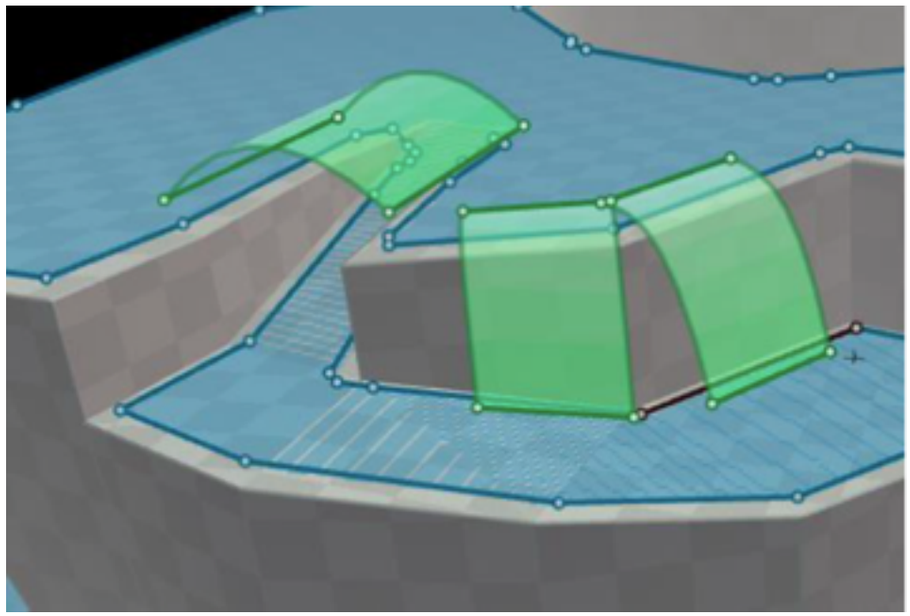
NavMeshes | POD (Points of discussion)

- Global Vs. Local Motion
- Static Avoidance Vs. Reactive(Dynamic) NavMesh
- Dynamic Avoidance
- Multiple NavMesh Vs. One Big NavMesh
- Finding the search node.

NavMeshes | Advantages and Disadvantages

- **Add information to the path**
- **Support for multi layered environments**
- **Virtual Worlds**
- **Special Regions encoded**
- **Automatic Generation**

- **Not really useful for real world scenarios**
- **Mesh generation**
- **Fine tuning needed to avoid corner cases**
- **Dynamic environments**
- **Wrongly designed NavMesh**



NavMeshes | *Conclusion*

- 3D Movement can be made without further complexity
- We are able to re-use familiar 2D algorithms with this approach
- Reduces the complexity of search space
- Enables moment freedom
- Must be built by hand or generated from previous world data

NavMeshes | Questions

Thank you very much!

Any further questions?

NavMeshes | References

[1] Y. Cui and G. Qin,"Intelligent path planning in 3d scene,"*International Conference on Computer Application and System Modelling*, 2010.

[2] M. A. DeLoura, Ed., *Game Programming Gems*. Charles River Media, 2000, vol. 1.

[3] L. Fischer and L. Nedel, "Semi-automatic navigation on 3d triangle meshes using bvp based path-planning," *SIBGRAPI Conference on Graphics, Patterns and Images*, 2011.

[4] S. Golodetz, "Automatic navigation mesh generation in configuration space."

[5] M. Kallmann, "Navigation queries from triangular meshes," *MIG*, vol. LNCS, no. 6459, pp. 230–241, 2010.

[6] H. Kim, K. Yu, and J. Kim, "Reducing the search space for pathfinding in navigation meshes by using visibility tests," *Journal of Electrical Engineering and Technology*, vol. 6, no. 6, p. 867 873, 2011.

[7] J. C. O'Neill, "Efficient navigation mesh implementation," *Journal of Game Development*, vol. 939, no. 9, pp. 71–90, March 2004.

[8] S. Rabin, Ed., AI Game Programming Wisdom. Charles River Media, 2006, vol. 3.

[9] *W. van Toll, A. F. C. IV, and R. Geraerts, "Navigation meshes for realistic multi-layered environments," IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011.*

[10] Tozour, Paul, "Building a Near-Optimal Navigation Mesh", *AI Game Programming Wisdom*, Ed. Steve Rabin, Charles River Media, 2002

[11] F. Farnstrom, "Improving on Near-Optimality: More Techniques for Building Navigation Meshes", *AI Game Programming Wisdom 3*, Ed. Steve Rabin, Charles River Media, 2006