Geometric and Discrete Path Planning for Interactive Virtual Worlds

Marcelo Kallmann
University of California Merced
mkallmann@ucmerced.edu
UCMERCED
UNIVERSITY OF CALIFORNIA, MERCED

Mubbasir Kapadia
Rutgers University
Mubbasir.kapadia@rutgers.edu

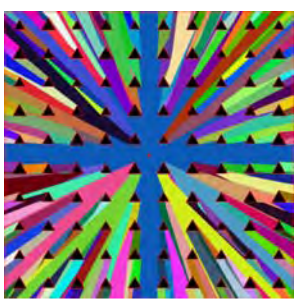
Introduction

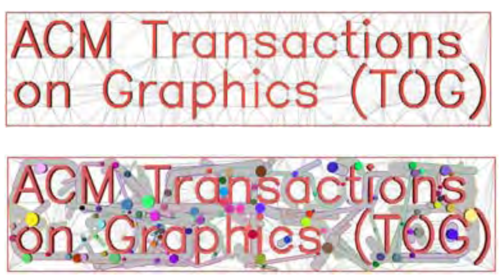
- Topics
  - Overview of the classical Computational Geometry and AI algorithms related to path planning
  - Overview of recent advances in planning methods for interactive virtual environments

Course Topics

1) Discrete and Geometric Planning (Marcelo,30min)

– A*, Shortest Paths, Visibility Graphs, Dijkstra, Shortest Path Maps, Navigation Meshes



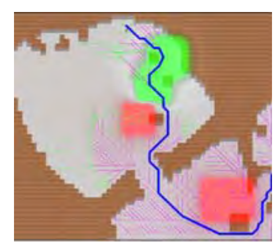


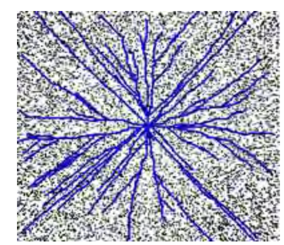
Examples: the shortest path map (left) and local clearance triangulation (right)

Course Topics

2) Advanced Planning Techniques (Mubbasir,20min)

– Extending classical A* to real-time constraints and dynamic scenarios, navigation with constraints, using GPU to speed up computations

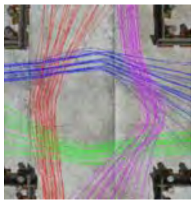


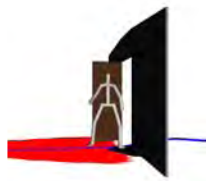


Course Topics

3) Planning for Character Animation
(Mubbasir and Marcelo, 30min)

– Character navigation problems, full-body and behavior planning, interactive narrative, etc.





Course: Modules

- Introduction (3 min)
- Discrete and Geometric Planning (Marcelo) (30min)
- Advanced Planning Techniques (Mubbasir) (20min)
- Planning for Animation (Mubbasir and Marcelo) (30min)
- Questions and Discussion (7min)

(We will take quick questions after each part as well)

Additional Information

- We will cover a lot of material in little time
  - Most topics will be covered as an overview
- Additional Material
  - SIGGRAPH course notes
  - Webpages of the authors:
    http://graphics.ucmerced.edu/
    http://www.cs.rutgers.edu/~mubbasir/
  - Recent book published by the authors:

Geometric and Discrete Path Planning for Interactive Virtual Worlds
Morgan & Claypool, 2016

Render the Possibilities SIGGRAPH2016

Module I
Discrete and Geometric Planning

Marcelo Kallmann
mkallmann@ucmerced.edu

UCMERCED
UNIVERSITY OF CALIFORNIA, MERCED

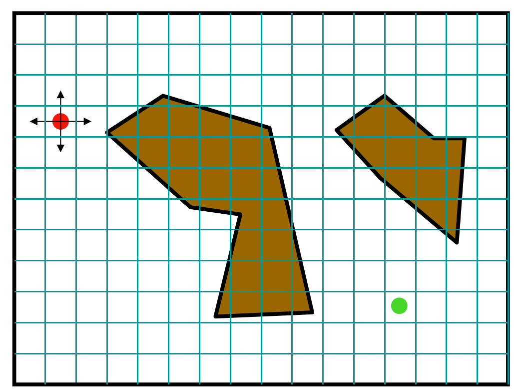
Geometric Path Planning

Introduction to Discrete Search

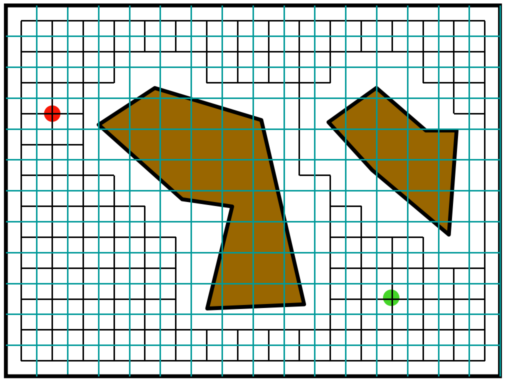
Discrete Search

- Main classical algorithms
  - Dijkstra
    - Search expansion outwards from source
  - A*
    - Reduces the number of nodes expanded with the use of a heuristic function
  - Both can be applied to generic graphs
    - Positive edge weights only
    - 4- or 8-connected grids are also graphs

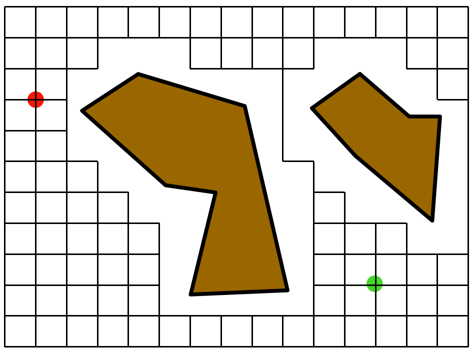
Ex: 4-Connected Grid Discretization



Equivalent to a Graph

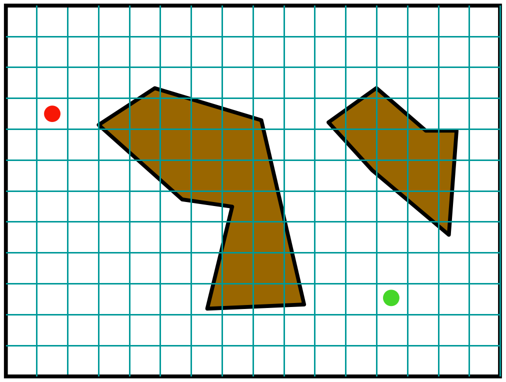


Equivalent to a Graph

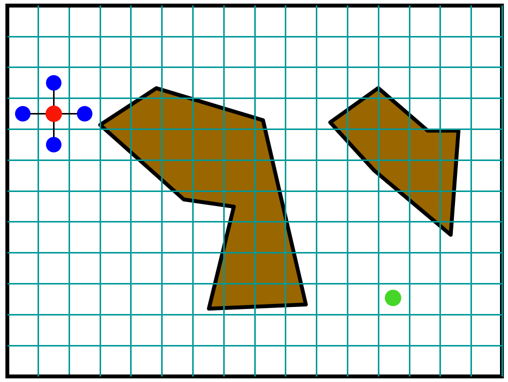


Example in Grid Discretization

- Example in a grid



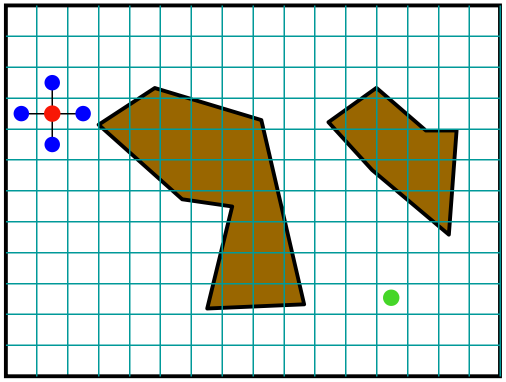
Example in Grid Discretization



Example in Grid Discretization

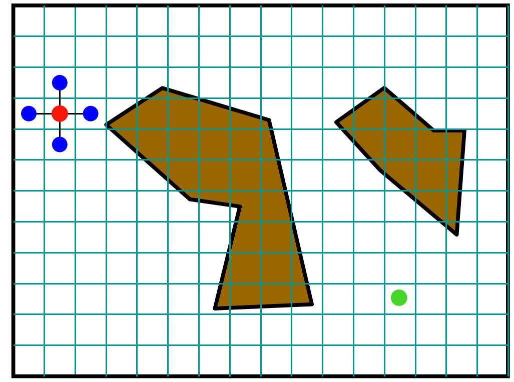
Q:

1 1 1 1



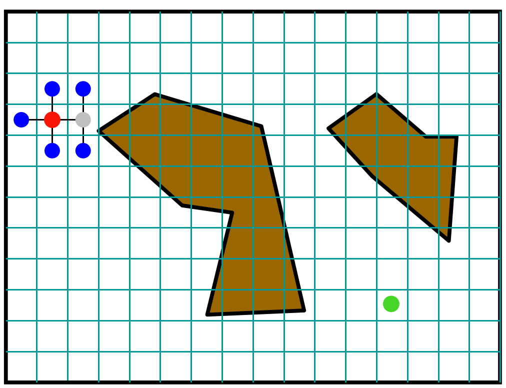
Example in Grid Discretization

1 1 1 1



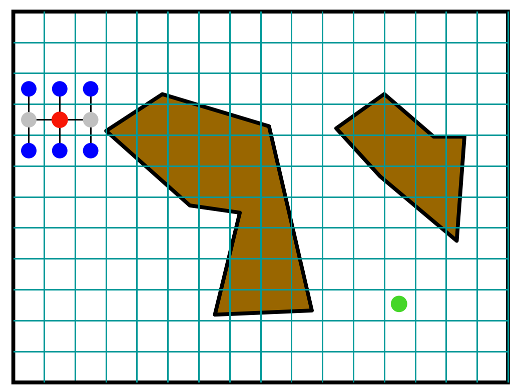
Example in Grid Discretization

1 1 1 2 2



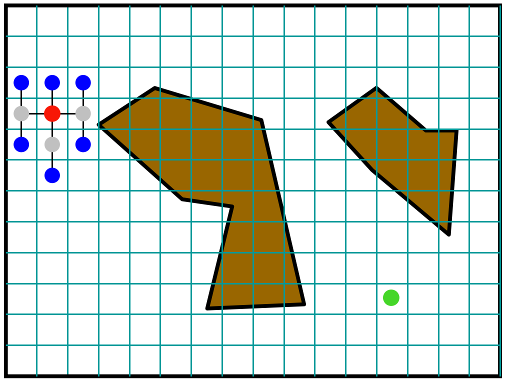
Example in Grid Discretization

1 1 2 2 2 2



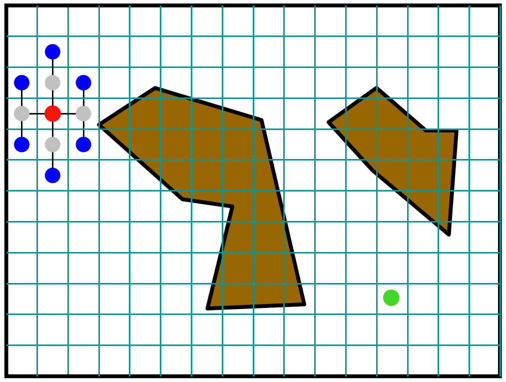
Example in Grid Discretization

1 2 2 2 2 2



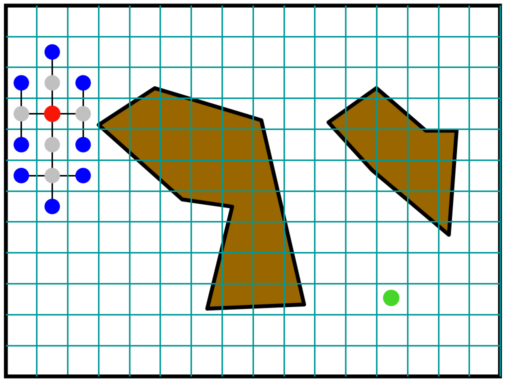
Example in Grid Discretization

2 2 2 2 2 2



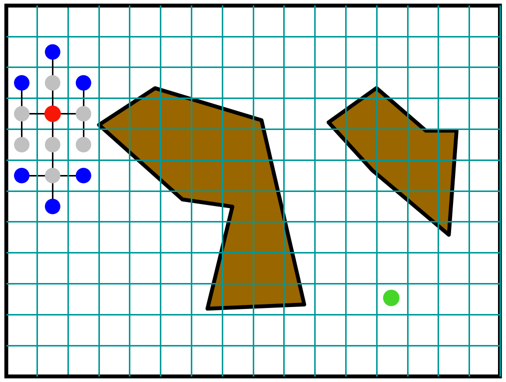
Example in Grid Discretization

2 2 2 2 2 3 3 3

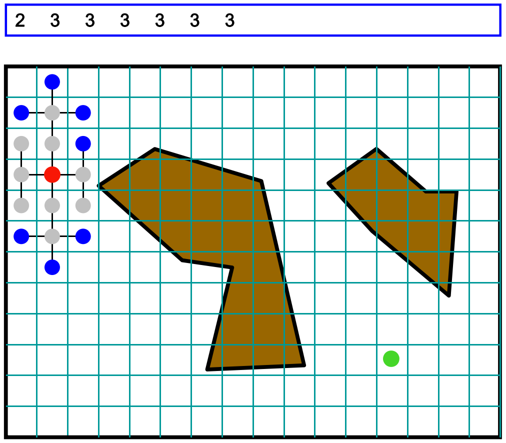


Example in Grid Discretization

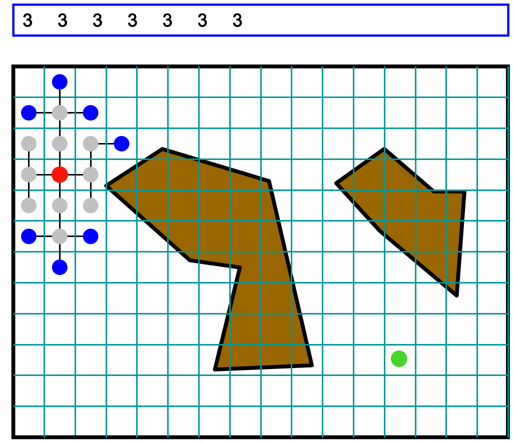
2 2 2 3 3 3



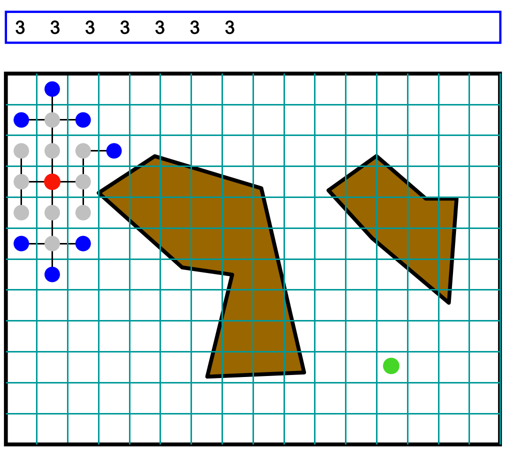
Example in Grid Discretization



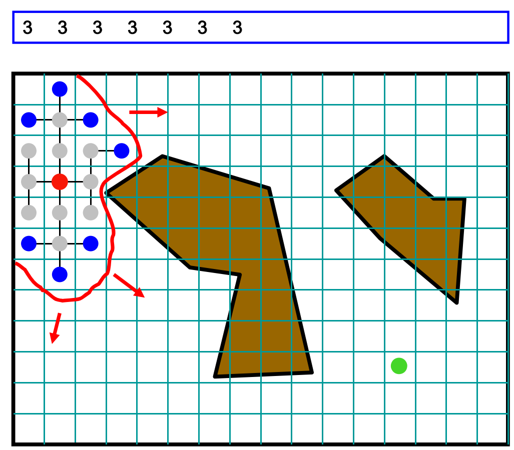
Example in Grid Discretization



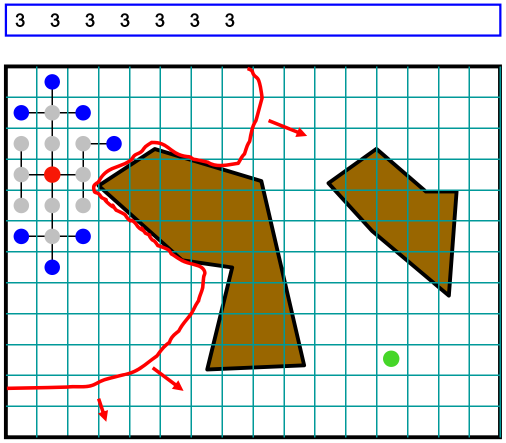
Example in Grid Discretization



Example in Grid Discretization

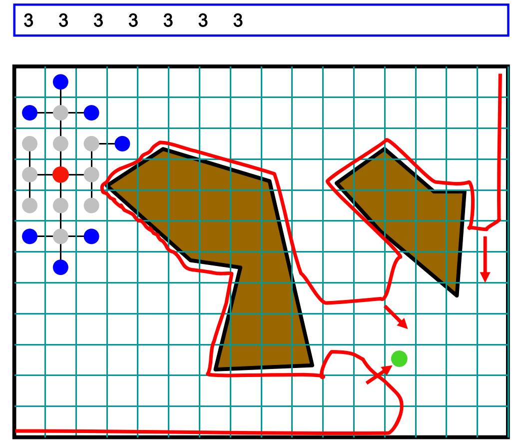


Example in Grid Discretization



wave front propagation
…

Example in Grid Discretization



wave front propagation
…

Algorithm: Dijkstra

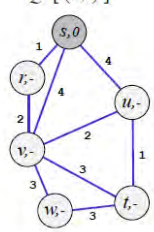
- Initialization

Algorithm 1 - Dijkstra Algorithm for Shortest Paths

Input: source node s and goal node t.
Output: shortest path from s to t, or null path if it does not exist.

Dijkstra(s, t)
Initialize Q with $(s, 0)$, set $g(s)$ to be 0, and mark s as visited;
while (Q not empty) **do**
  $v \leftarrow Q.remove()$;
  if ($v = t$) **return** reconstructed branch from v to s;
  for each (neighbors n of v) **do**
    if (n not visited **or** $g(n) > g(v) + c(v, n)$) **then**
      Set the parent of n to be v;
      Set $g(n)$ to be $g(v) + c(v, n)$;
      if (n visited) $Q.decrease(n, g(n))$; **else** $Q.insert(n, g(n))$;
      Mark n as visited, if not already visited;
    end if
  end for
end while
return null path;

Q: [(0;*s*)]



Algorithm: Dijkstra

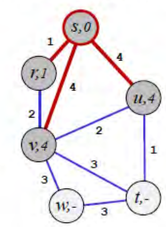
- Iteration 1: all neighbors go to *Q*

Algorithm 1 - Dijkstra Algorithm for Shortest Paths

Input: source node s and goal node t.
Output: shortest path from s to t, or null path if it does not exist.

Dijkstra(s, t)
Initialize Q with $(s, 0)$, set $g(s)$ to be 0, and mark s as visited;
while (Q not empty) **do**
  $v \leftarrow Q.remove()$;
  if ($v = t$) **return** reconstructed branch from v to s;
  for each (neighbors n of v) **do**
    if (n not visited **or** $g(n) > g(v) + c(v, n)$) **then**
      Set the parent of n to be v;
      Set $g(n)$ to be $g(v) + c(v, n)$;
      if (n visited) $Q.decrease(n, g(n))$; **else** $Q.insert(n, g(n))$;
      Mark n as visited, if not already visited;
    end if
  end for
end while
return null path;

1) Q: [(1;*r*), (4;*u*), (4;*v*)]



Algorithm: Dijkstra

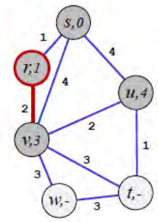
- Iteration 2: decrease key called for *v*

Algorithm 1 - Dijkstra Algorithm for Shortest Paths

Input: source node s and goal node t.
Output: shortest path from s to t, or null path if it does not exist.

```
Dijkstra(s, t)
Initialize Q with (s, 0), set g(s) to be 0, and mark s as visited;
while ( Q not empty ) do
  v ← Q.remove();
  if ( v = t ) return reconstructed branch from v to s;
  for each ( neighbors n of v ) do
    if ( n not visited or g(n) > g(v) + c(v, n) ) then
      Set the parent of n to be v;
      Set g(n) to be g(v) + c(v, n);
      if ( n visited ) Q.decrease(n, g(n)); else Q.insert(n, g(n));
      Mark n as visited, if not already visited;
    end if
  end for
end while
return null path;
```

2) Q: [(3;*v*), (4;*u*)]



Algorithm: Dijkstra

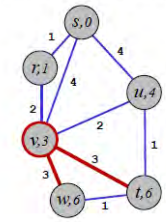
- Iteration 3: target node *t* goes to *Q*

Algorithm 1 - Dijkstra Algorithm for Shortest Paths

Input: source node s and goal node t.
Output: shortest path from s to t, or null path if it does not exist.

```
Dijkstra(s, t)
Initialize Q with (s, 0), set g(s) to be 0, and mark s as visited;
while ( Q not empty ) do
  v ← Q.remove();
  if ( v = t ) return reconstructed branch from v to s;
  for each ( neighbors n of v ) do
    if ( n not visited or g(n) > g(v) + c(v, n) ) then
      Set the parent of n to be v;
      Set g(n) to be g(v) + c(v, n);
      if ( n visited ) Q.decrease(n, g(n)); else Q.insert(n, g(n));
      Mark n as visited, if not already visited;
    end if
  end for
end while
return null path;
```

3) Q: [(4;*u*), (6;*w*), (6;*t*)]



Algorithm: Dijkstra

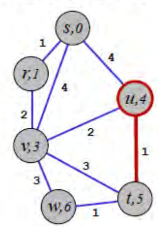
- Iteration 4: decrease key called for *t*

Algorithm 1 - Dijkstra Algorithm for Shortest Paths

Input: source node s and goal node t.
Output: shortest path from s to t, or null path if it does not exist.

```
Dijkstra(s, t)
Initialize Q with (s, 0), set g(s) to be 0, and mark s as visited;
while ( Q not empty ) do
  v ← Q.remove();
  if ( v = t ) return reconstructed branch from v to s;
  for each ( neighbors n of v ) do
    if ( n not visited or g(n) > g(v) + c(v, n) ) then
      Set the parent of n to be v;
      Set g(n) to be g(v) + c(v, n);
      if ( n visited ) Q.decrease(n, g(n)); else Q.insert(n, g(n));
      Mark n as visited, if not already visited;
    end if
  end for
end while
return null path;
```

4) Q: [(5;*t*), (6;*w*)]



Algorithm: Dijkstra

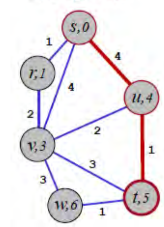
- Iteration 5

Algorithm 1 - Dijkstra Algorithm for Shortest Paths

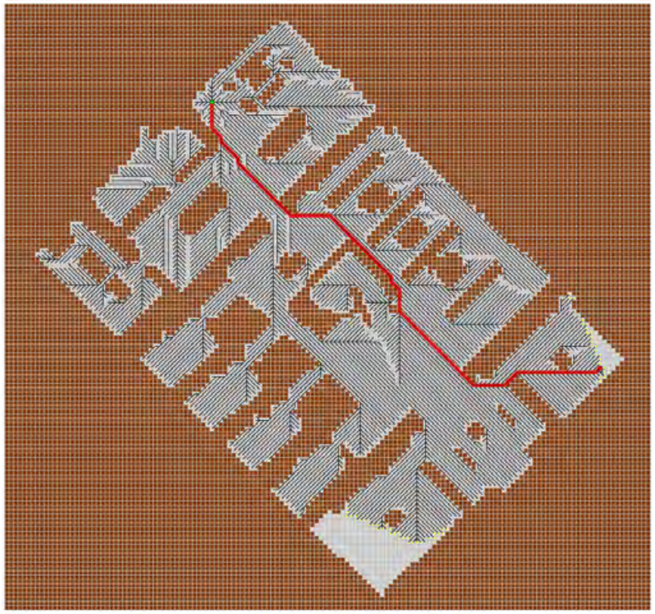
Input: source node s and goal node t.
Output: shortest path from s to t, or null path if it does not exist.

```
Dijkstra(s, t)
Initialize Q with (s, 0), set g(s) to be 0, and mark s as visited;
while ( Q not empty ) do
  v ← Q.remove();
  if ( v = t ) return reconstructed branch from v to s;
  for each ( neighbors n of v ) do
    if ( n not visited or g(n) > g(v) + c(v, n) ) then
      Set the parent of n to be v;
      Set g(n) to be g(v) + c(v, n);
      if ( n visited ) Q.decrease(n, g(n)); else Q.insert(n, g(n));
      Mark n as visited, if not already visited;
    end if
  end for
end while
return null path;
```

5) Q: [(6;*w*)]



Example



Algorithm: A*

- Includes Heuristic
  - Cost becomes cost-to-come + cost-to-go
  - Typical cost-to-go heuristic: dist(node,goal)

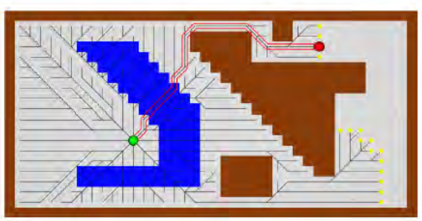
Algorithm 2 - A* Algorithm for Shortest Paths

Input: source node s and goal node t.
Output: shortest path from s to t, or null path if it does not exist.

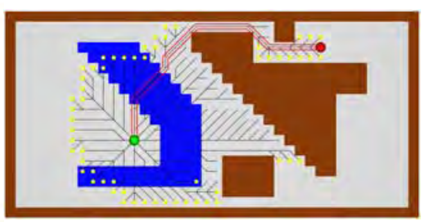
AStar(s, t)
Initialize Q with $(s, 0)$, set $g(s)$ to be 0, and mark s as visited;
while (Q not empty) **do**
  $v \leftarrow Q.remove()$;
  if ($v = t$) **return** reconstructed branch from v to s;
  for all (neighbors n of v) **do**
    if (n not visited **or** $g(n) > g(v) + c(v, n)$) **then**
      Set the parent of n to be v;
      Set $g(n)$ to be $g(v) + c(v, n)$;
      if (n visited) **then** $Q.decrease(n, g(n) + h(n))$;
      else $Q.insert(n, g(n) + h(n))$;
      Mark n as visited, if not already visited;
return null path;

Example

Dijkstra



A*



Analysis

- Priority Queue
  - Self-balancing binary tree or a binary min-heap
    - Insertion, removal and decrease: O(log(k))
  - Simplifications possible
    - Decrease operation not as simple to implement
    - Good option: to “insert again” instead of a decrease
- Overall time
  - O ((n+m) log n)
    (n = number of vertices, m = number of edges)
  - Equivalent to O (m log n)
    - Note: m may be $O(n^2)$

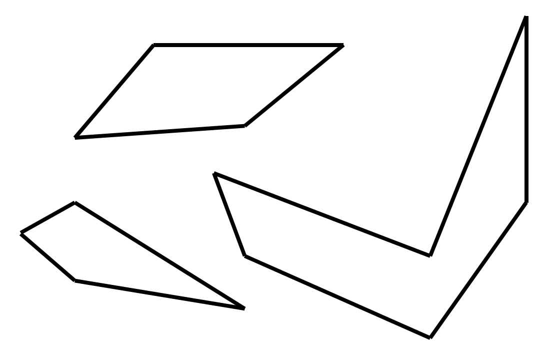
Euclidean Shortest Paths (ESPs)

- Shortest paths in the Euclidean plane
  - Paths are “globally” shortest in the plane
    - And not in a given graph representing the plane
    - Cannot be efficiently reduced to a simple graph search
  - Most popular method
    - Search the “Visibility Graph”
      - unfortunately it has O(n^2) edges (n = # obs vertices)
  - But it can be computed in O(n log n)
    - Using the “continuous Dijkstra” approach
      - Optimal algorithm difficult to implement in practice
      - More about that later

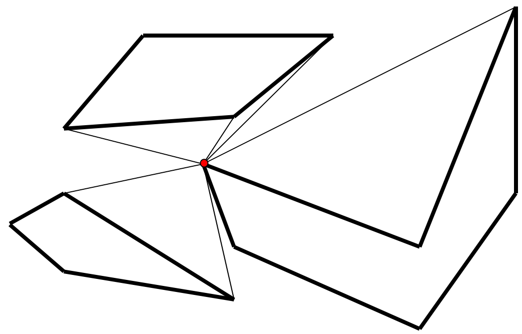
Visibility Graph

Visibility Graph

- Edges connect all pairs of visible vertices

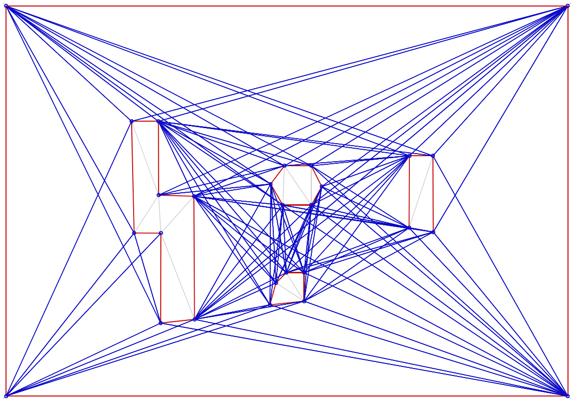


Visibility Graph



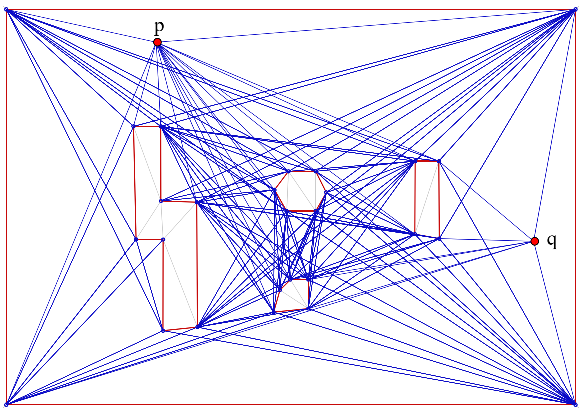
Visibility Graph

- It can be preprocessed
  - Query points added later at run-time



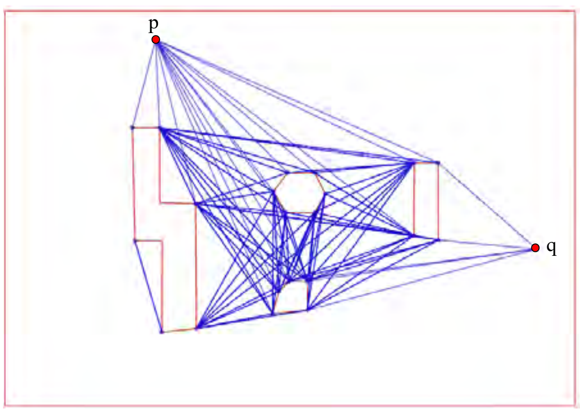
Visibility Graph

- Full visibility graph
  - Optimizations are possible



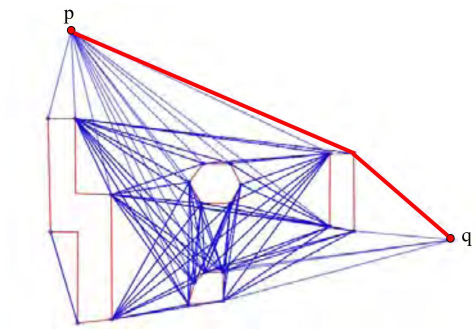
Visibility Graph

- Optimizations are possible
  - Ex: discard edges connecting “concave corners”



Visibility Graph

- Final graph for path search
  - Ready for a discrete path search algorithm



  - Shortest path in the Visibility Graph is the ESP

Visibility Graph

- Preprocessing for a specific clearance value
  - Lozano-Pérez and Wesley 1979
  - Chew 1985
    - First dilates the environment, then computes visibility graph of tangents
      - Pre-computation: O(n^2 log n), size: O(n^2), query: O(n^2 log n)
- Clearance-independent preprocessing possible
  - Wein, van den Berg and Halperin, "***the visibility–Voronoi complex and its applications***", 2007
    - Preprocessing: O (n^2 log n)
    - Query time: O *(n* log *n + m) = O (n^2)*
    - *Probably the best practical method for global optimality*

Pre-processing for a source point:

Shortest Path Tree

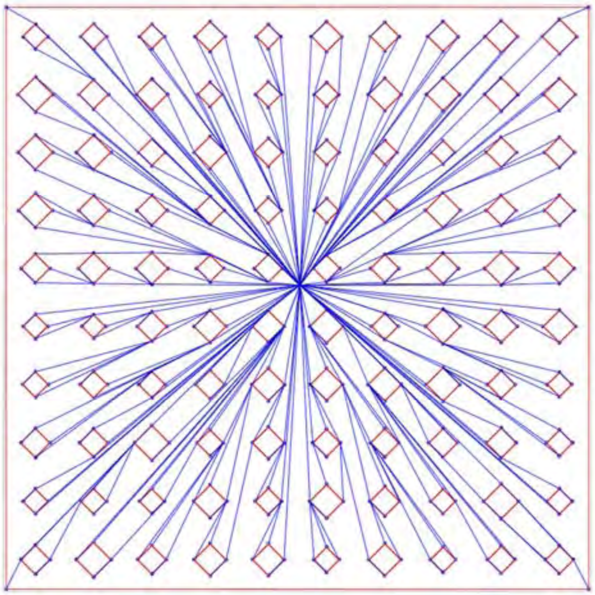
The Shortest Path Tree

- Contains shortest paths from all vertices to source point
  - Can be computed from the visibility graph with an exhaustive Dijkstra Expansion

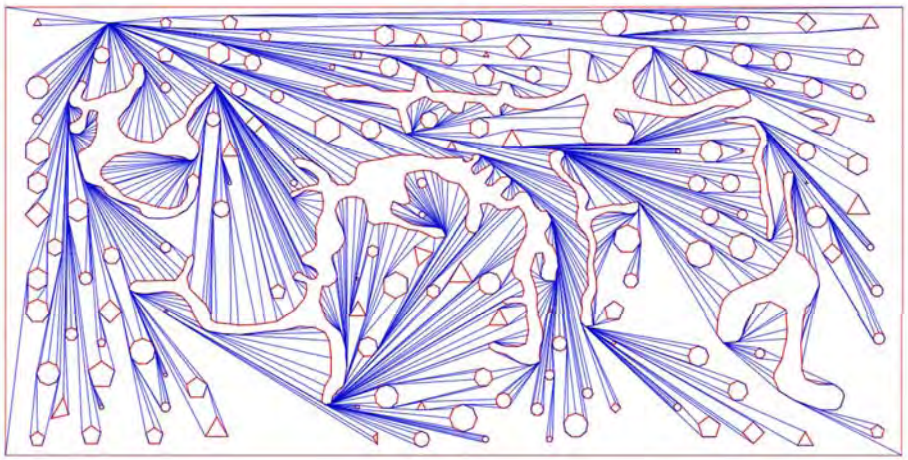
Algorithm 2 **Dijkstra SPT Expansion**

```
function BUILDSPT ( p )
    Initialize priority queue Q with p;
    Mark node of p as visited;
    while ( Q not empty ) do
        s ← Q.remove();
        for all ( neighbors n of s ) do
            if (n not visited or g(n) > g(s) + d(s, n)) then
                Set the SPT parent of n to be s;
                Set g(n) to be g(s) + d(s, n);
                Insert n with cost g(n) in Q;
                Mark n as visited;
```

The Shortest Path Tree: Example



The Shortest Path Tree: Example



The Shortest Path Tree

- The SPT is rooted at some source point
- Given a destination point, how to use the SPT ?
  - First compute visible vertices V to query point
  - Identify vertex $v \in V$ that is in the shortest path to source point
    - Simple given that vertices store their geodesic distances to the SPT source (cost *g*)
  - Shortest path is branch passing by v

Continuous Dijkstra

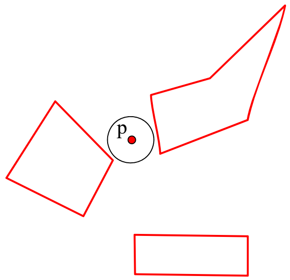
Continuous Dijkstra

- Addresses the whole plane
- Principle is the same as discrete SPT
  - But is continuous, will generate a Shortest Path Map (SPM) partition of the plane in O(*n*) cells
    - Represents all shortest paths from the source to any point in the continuous plane
    - Once the SPM is computed, ESPs to the source point can be efficiently computed
  - It is based on the simulation of a "continuous wavefront propagation" from the source point

[Mitchell 1991; Mitchell 1993], [Hershberger and Suri 1997]

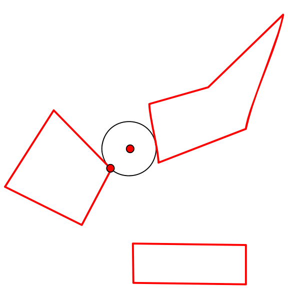
Continuous Dijkstra

- Wavefront propagation
  - Every point in the wavefront border has equal distance to the source point *p*



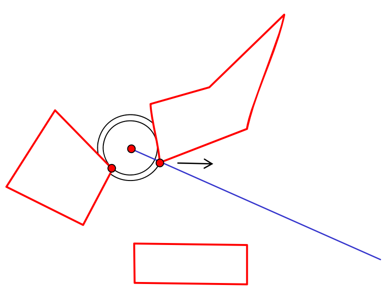
Continuous Dijkstra

- Wavefront propagation
  - Vertices hit by the wavefront will be visible to their wave generators



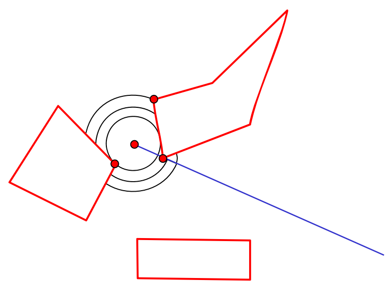
Continuous Dijkstra

- Wavefront propagation
  - Every time a vertex is reached, a new wave generator will cover the unseen region from the previous generator



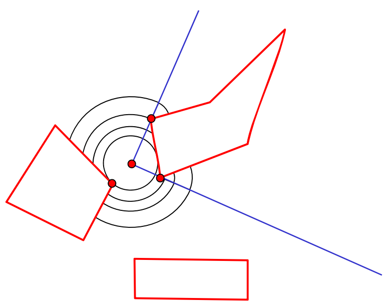
Continuous Dijkstra

- Wavefront propagation
  - New vertices are processed as they are reached



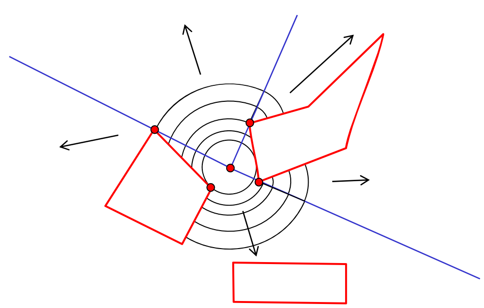
Continuous Dijkstra

- Wavefront propagation
  - New vertices are processed as they are reached



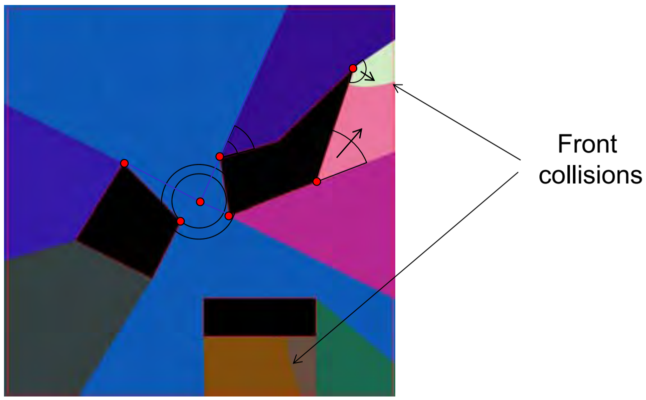
Continuous Dijkstra

- Wavefront propagation
  - All points in the wavefront border remain with equal geodesic distance to the source point



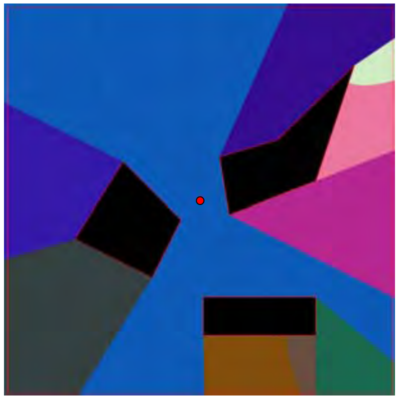
Continuous Dijkstra

- Front will eventually collide with itself forming hyperbolic frontiers



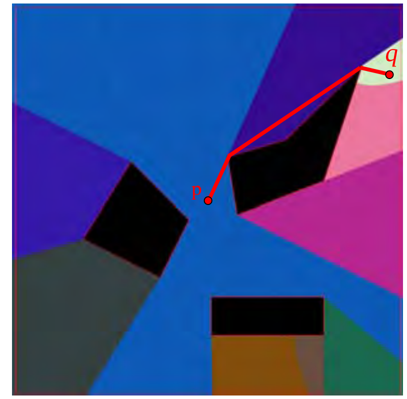
Continuous Dijkstra

- Result: Shortest Path Map
  - Captures all possible shortest paths to the source point



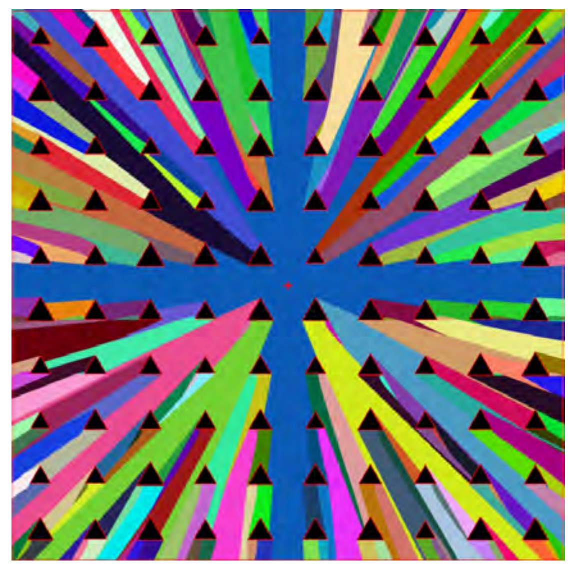
Continuous Dijkstra

- Path extraction from SPM
  - First find region containing goal point, then trace back generator vertices

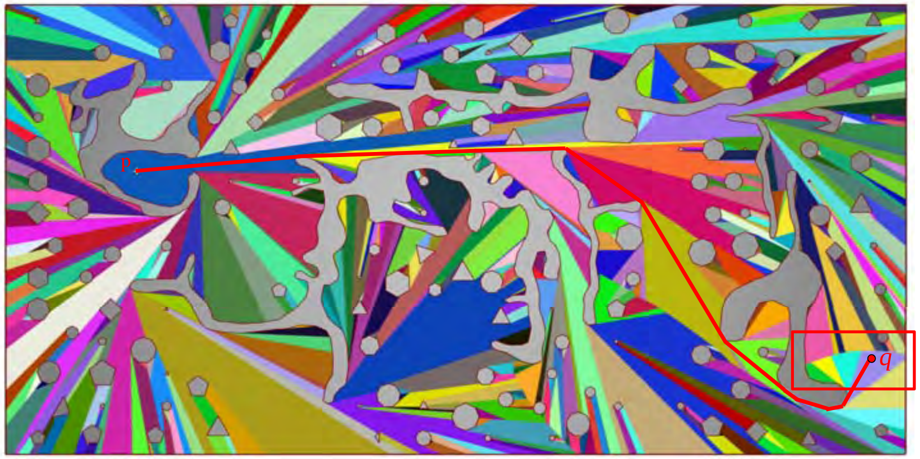


Continuous Dijkstra: Example

Continuous Dijkstra: Example

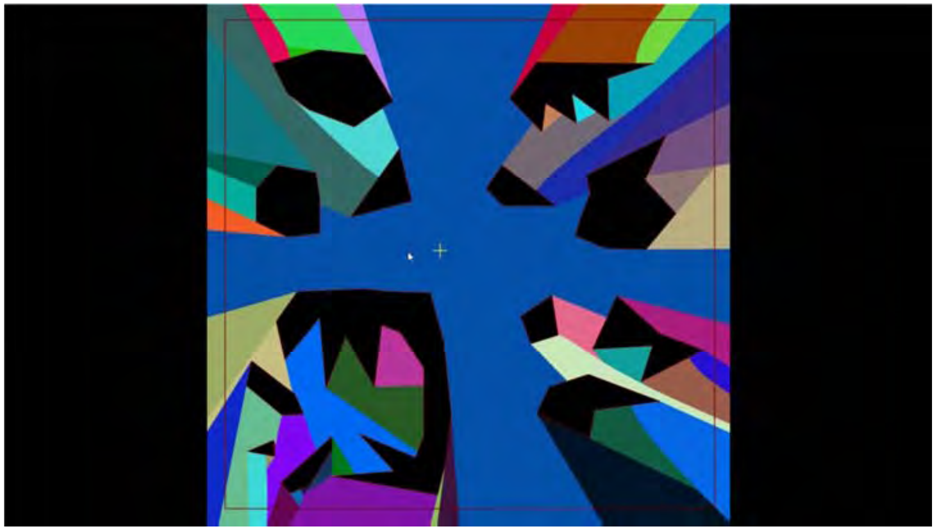


Continuous Dijkstra: Example



Camporesi and Kallmann, Computing Shortest Path Maps with GPU Shaders, MIG 2014.

Continuous Dijkstra: Example



Camporesi and Kallmann, Computing Shortest Path Maps with GPU Shaders, MIG 2014.

Continuous Dijkstra: Extensions

(work in preparation)

Additional Geometric Representations useful for Path Planning

http://graphics.ucmerced.edu/

Navigation Meshes

Navigation Meshes

- Navigation meshes are a representation of the free environment
  - For virtual worlds, being fast is most important
    - Computing ESPs is usually not addressed
- What properties should we expect?

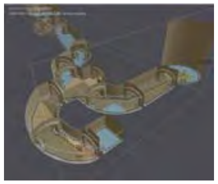
Summary of Expected Properties

- Linear number of cells
  - Critical for path search to run in optimal times
- Quality of paths
  - Locally shortest paths should be provided
- Arbitrary clearance
  - Same structure should handle any clearance value
- Representation robustness
  - Intersections, overlaps, etc. should be handled
- Dynamic updates
  - Efficient updates when environment changes

Approaches

- Many approaches are possible
  - Coarser cell decompositions possible (less nodes to search)
    - Ex.: NEOGEN [Oliva and Pelechano 2013]

  - Complete solutions for path planning have been developed
    - Ex.: Recast & Detour toolkit, freely available

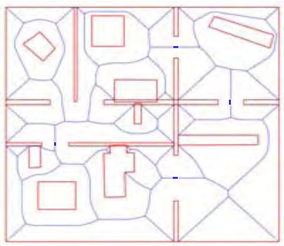


  - However meshes need to be pre-processed for each given desired clearance

Approaches

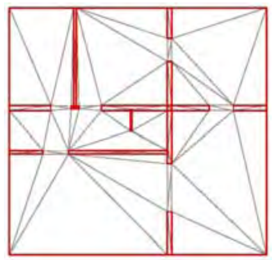
- Structures most suitable for handling arbitrary clearance efficiently:

Medial Axis



Medial axis represents paths of maximum clearance

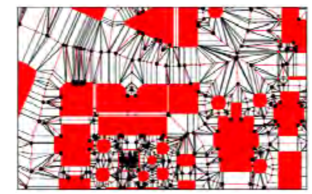
CDTs



CDT decomposes the free space in O(*n*) triangles

Medial Axis

- Medial Axis as a navigation mesh
  - Good amount of work available
    - For ex.: extensions for multi layered environments and for handling dynamic updates available
      - Geraerts, “Planning Short Paths with Clearance using Explicit Corridors”, 2010
      - van Toll et al., “Navigation Meshes for Realistic Multi-Layered Environments”, 2011
      - van Toll et al., “A Navigation Mesh for Dynamic Environments”, 2012



Triangulations

- Triangulations as navigation meshes
  - Triangle meshes are relatively simple to build
    - Are composed of only straight edges
  - Paths can be easily computed
    - However handling clearance is not straightforward
  - Can easily generate locally shortest paths
    - For instance corridors will be already triangulated and ready for the Funnel algorithm

(recent benchmark work shows that triangulations are faster)

Triangulations

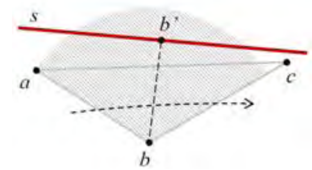
- However, clearance not directly represented
  - Clearance checks per edge not enough
    - Even if additional free edges are inserted to improve capturing clearance in corridors
      [Lamarche and Donikian, “Crowd of Virtual Humans: a New Approach for Real Time Navigation in Complex and Structured Environments”, 2004]
  - Clearance checks per triangle not enough
    - Previous attempts do not always work
      [Demyen and Buro, “Efficient triangulation-based pathfinding”, 2006]

Triangulations

- Local Clearance Triangulations (LCTs)
  - Proposes a refinement strategy for CDTs allowing clearance information to be stored in the triangulation
  - Details in TOG 2014
    - Kallmann, "Dynamic and Robust Local Clearance Triangulations", 2014

Local Clearance Triangulations

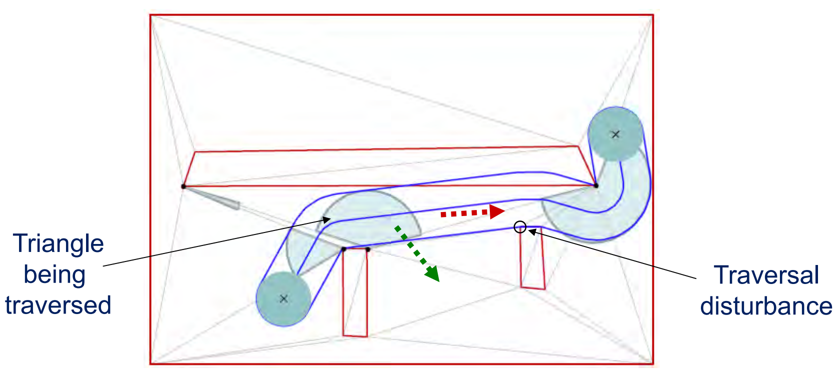
- Clearance Defined per triangle traversal
  - Traversal from ab to bc: τ_{abc}



  - Traversal clearance: $cl(a,b,c) = dist(b,s)$
    s is the constraint *behind* ac and closest to b

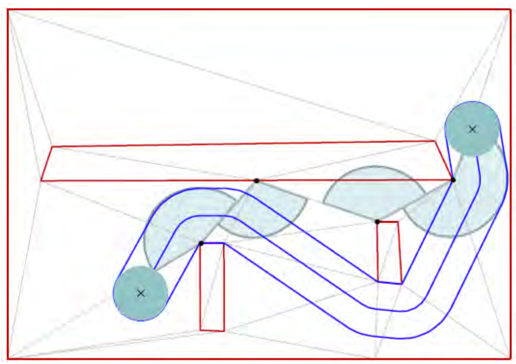
Local Clearance Triangulations

- However clearance metric not enough…
  - Clearance in the red arrow direction not well captured



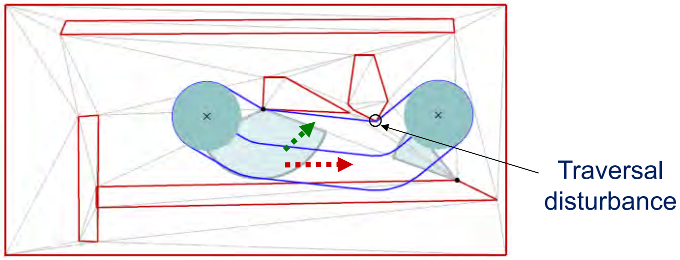
Local Clearance Triangulations

- But it can work if there are no disturbances
  - By refining the triangulation disturbances can be eliminated and correct paths are obtained



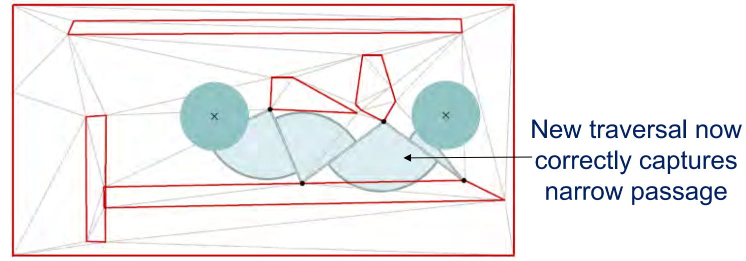
Local Clearance Triangulations

- Refinements solve disturbances
  - Disturbances appear when a traversal does not correctly captures the local clearance of all possible exit directions



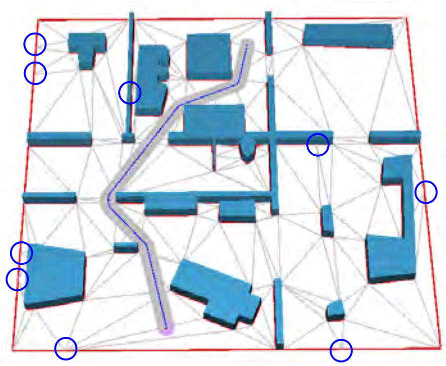
Local Clearance Triangulations

- Refinements solve disturbances
  - Now all disturbances have been eliminated with refinements
  - Correct result: no valid path exists

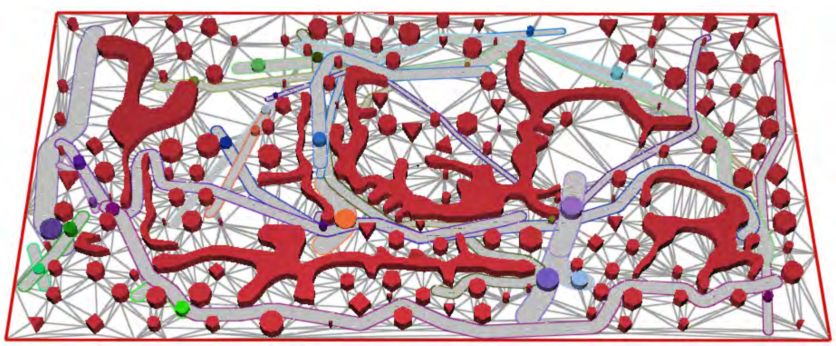


Local Clearance Triangulations

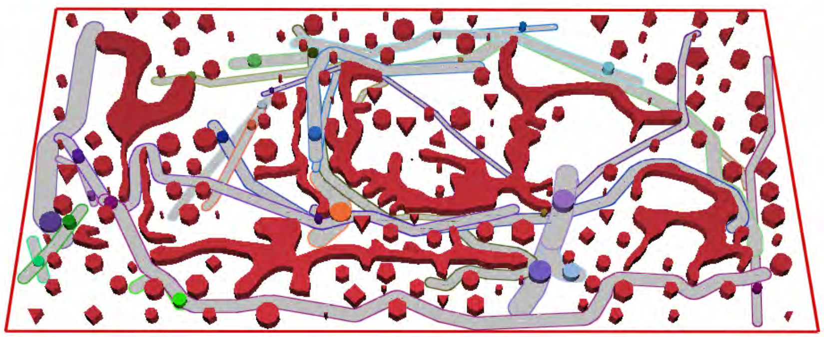
- Example of refinements
  - Total number of vertices remain O(n)



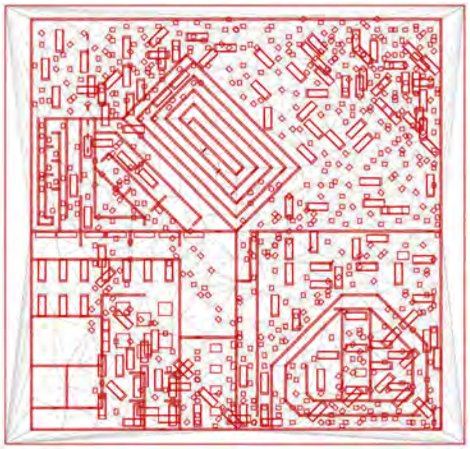
Example LCT



Example LCT

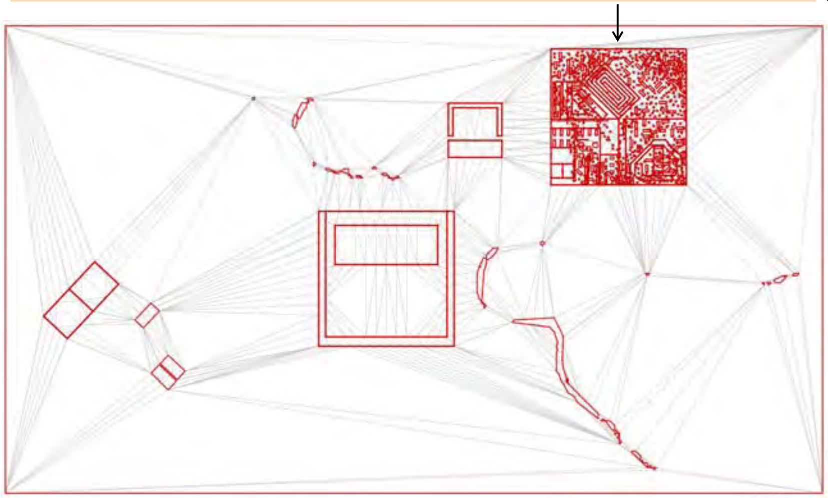


Example LCT



Test environment for The Sims 4: each small square represents a static character, later dynamically removed when it is time to walk
[used with permission]

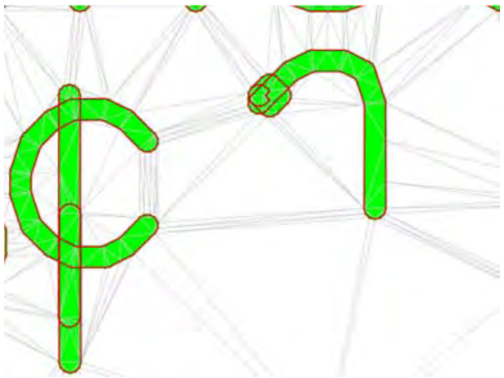
Example



Efficiently representation of environments at different scales

New Results on LCTs

- **Dynamic Operations** with management of refinements
- **Robust operations** addressing self-intersections at run-time



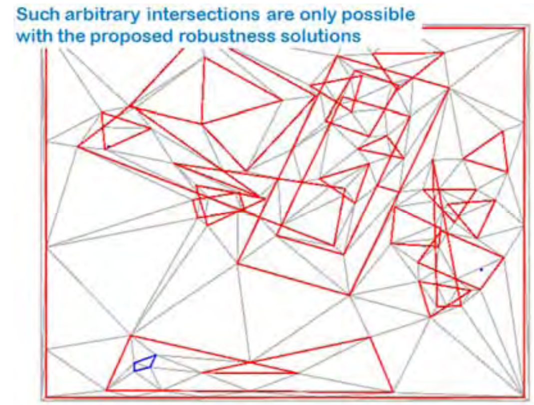
M. Kallmann, “Dynamic and Robust Local Clearance Triangulations”, TOG 2014

Dynamic Updates: Example

– Dynamic updates while maintaining the mesh ready for arbitrary clearance path queries

Robustness: Example

– Robust watertight dynamic updates at run-time



– Robustness with floating point representation is achieved with one exact point location test for correctness detection and perturbation of invalid coordinates

Summary

- Euclidean Shortest Paths are difficult to be computed efficiently
  - Visibility Graph popular but is a O(n^2) structure
  - Continuous Dijkstra methods promising
- Navigation Meshes
  - Focus on efficient path planning
  - Medial axis gives paths of maximum clearance
  - Triangulations can be used to efficiently compute paths with arbitrary clearance

Questions?

Render the Possibilities
SIGGRAPH2016

Advanced Planning Techniques

Mubbasir Kapadia

www.cs.rutgers.edu/~mubbasir

Challenges

Real-time Planning in Dynamic Environments

Planning with Constraints

Scaling to large worlds and many agents

Proposed Solutions

~~Real-time Planning in Dynamic Environments~~

From Classical A* to Anytime Dynamic Search

Planning with Constraints

Scaling to large worlds and many agents

Proposed Solutions

~~Real-time Planning in Dynamic Environments~~

From Classical A* to Anytime Dynamic Search

~~Planning with Constraints~~

Constraint-Aware Navigation in Dynamic Environments

Scaling to large worlds and many agents

Proposed Solutions

~~Real-time Planning in Dynamic Environments~~

From Classical A* to Anytime Dynamic Search

~~Planning with Constraints~~

Constraint-Aware Navigation in Dynamic Environments

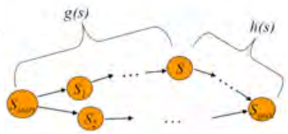
~~Scaling to large worlds and many agents~~

Anytime Dynamic Search on the GPU

A* Search Algorithm

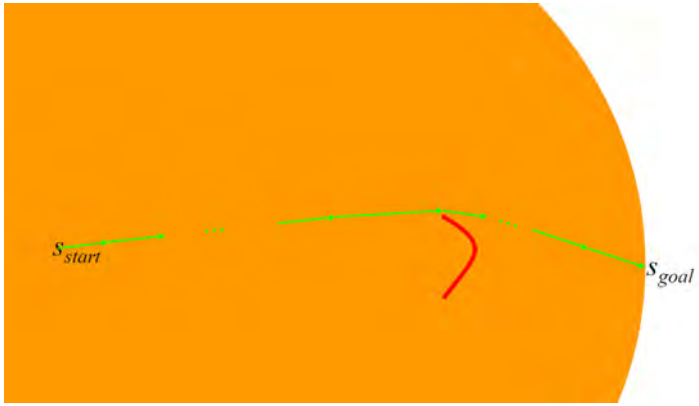
Computes optimal g-values of relevant states

procedure ComputePath()
while(s_{goal} is not expanded)
  remove s with the smallest $f(s)$ from *OPEN*;
  for each successor s' of s
    if $g(s') > g(s) + c(s, s')$
      $g(s') = g(s) + c(s, s')$;
      insert/update s' in *OPEN* with $f(s') = g(s') + h(s')$;



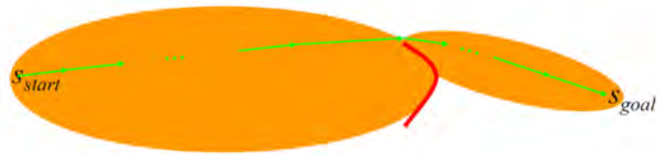
Dijkstra's Search Expansion

Expands state in the order of f = g *values*



A* Search Expansion

Expands state in the order of f = g+h *values*

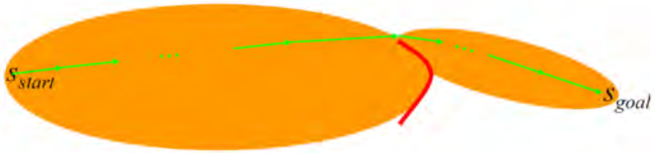


Courtesy Likhachev 2010

A* Search Expansion

Expands state in the order of f = g+h *values*

- For large problems, this results in A* quickly running out of memory

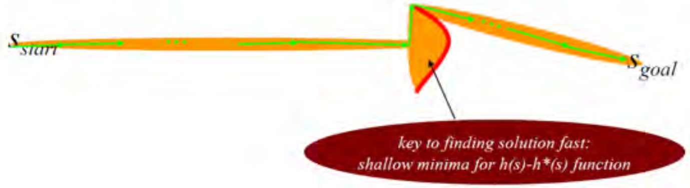


Courtesy Likhachev 2010

Weighted A* Search Expansion

Expands states in the order of f = g + ε.h *values*

- ε > 1 bias towards states that are closer to goal



Courtesy Likhachev 2010

Anytime Repairing A* (ARA*)

Efficient series of weighted A searches with decreasing ε*

set ε to large value;
$g(s_{start}) = 0$; v-values of all states are set to infinity;
while $\varepsilon \geq 1$
    CLOSED = {}; *INCONS* = {};
    ComputePathwithReuse();
    publish current ε suboptimal solution;
    decrease ε;
    initialize *OPEN* = *OPEN U INCONS*;

ARA*: Anytime A* with Provable Bounds on Sub-Optimality
Maxim Likhachev, Geoff Gordon and Sebastian Thrun
Advances in Neural Information Processing Systems, 2003

Anytime Repairing A* (ARA*)

initialize *OPEN* with all overconsistent states;
ComputePathwithReuse function
while($f(s_{goal})$ > minimum f-value in *OPEN*)
  remove s with the smallest $[g(s)+ \varepsilon h(s)]$ from *OPEN*;
  insert s into *CLOSED*;
  $v(s)=g(s)$;
  for every successor s' of s
    if $g(s') > g(s) + c(s,s')$
      $g(s') = g(s) + c(s,s')$;
      if s' not in *CLOSED* then insert s' into *OPEN*;
      otherwise insert s' into *INCONS*

Anytime Repairing A* (ARA*)

initialize *OPEN* with all overconsistent states;

ComputePathwithReuse function

Consistent State:

$$g(s') = \min_{s'' \in pred(s')}(g(s'') + c(s'', s')) = g(s) + c(s, s')$$

Inconsistent State:

$$g(s') > \min_{s'' \in pred(s')}(g(s'') + c(s'', s'))$$

if *s'* not in *CLOSED* then insert *s'* into *OPEN*;
otherwise insert *s'* into *INCONS*

www.cs.rutgers.edu/~mubbasir

Anytime Repairing A* (ARA*)

initialize *OPEN* with all overconsistent states;
ComputePathwithReuse function
while($f(s_{goal})$ > minimum f-value in *OPEN*)
remove *s* with the smallest $[g(s)+ \varepsilon h(s)]$ from *OPEN*;
insert *s* into *CLOSED*;
$v(s)=g(s)$;
for every successor *s'* of *s*
if $g(s') > g(s) + c(s,s')$
$g(s') = g(s) + c(s,s')$;
if *s'* not in *CLOSED* then insert *s'* into *OPEN*;
otherwise insert *s'* into *INCONS*

www.cs.rutgers.edu/~mubbasir

Anytime Repairing A* (ARA*)

initialize *OPEN* with all overconsistent states;
ComputePathwithReuse function
while($f(s_{goal})$ > minimum f-value in *OPEN*)
remove *s* with the smallest $[g(s)+ \varepsilon h(s)]$ from *OPEN*;
insert *s* into *CLOSED*;
$v(s)=g(s)$;
for every successor *s'* of *s*
if $g(s') > g(s) + c(s,s')$
$g(s') = g(s) + c(s,s')$;
if *s'* not in *CLOSED* then insert *s'* into *OPEN*;
otherwise insert *s'* into *INCONS*

www.cs.rutgers.edu/~mubbasir

Anytime D*

Combined properties of anytime and dynamic planning

Set ε to large value
While goal is not reached
    ComputePathWithReuse()
    Publish ε-suboptimal path
    Follow path until map is updated
    Update corresponding edge costs
    Set start to current state of agent
    If significant changes were observed
        Increase ε or replan from scratch
    Else
        Decrease ε

Anytime search in dynamic graphs
Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun
Journal of Artificial Intelligence, 2008

www.cs.rutgers.edu/~mubbasir

Anytime D*

Combined properties of anytime and dynamic planning

Set ε to large value
While goal is not reached
    ComputePathWithReuse()
    Publish ε-suboptimal path
    Follow path until map is updated
    Update corresponding edge costs
    Set start to current state of agent
    If significant changes were observed
        Increase ε or replan from scratch
    Else
        Decrease ε

Anytime search in dynamic graphs
Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun
Journal of Artificial Intelligence, 2008

www.cs.rutgers.edu/~mubbasir

Anytime D*

Combined properties of anytime and dynamic planning

Set ε to large value
While goal is not reached
    ComputePathWithReuse()
    Publish ε-suboptimal path
    Follow path until map is updated
    Update corresponding edge costs
    Set start to current state of agent
    If significant changes were observed
        Increase ε or replan from scratch
    Else
        Decrease ε

Anytime search in dynamic graphs
Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun
Journal of Artificial Intelligence, 2008

www.cs.rutgers.edu/~mubbasir

Anytime D*

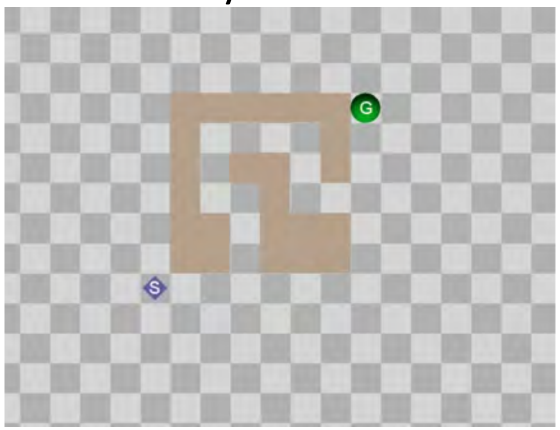
Combined properties of anytime and dynamic planning

Set ε to large value
While goal is not reached
    ComputePathWithReuse()
    Publish ε-suboptimal path
    Follow path until map is updated
    Update corresponding edge costs
    Set start to current state of agent
    If significant changes were observed
        Increase ε or replan from scratch
    Else
        Decrease ε

Anytime search in dynamic graphs
Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun
Journal of Artificial Intelligence, 2008

www.cs.rutgers.edu/~mubbasir

Anytime D*



www.cs.rutgers.edu/~mubbasir

Proposed Solutions

~~Real-time Planning in Dynamic Environments~~

From Classical A* to Anytime Dynamic Search

~~Planning with Constraints~~

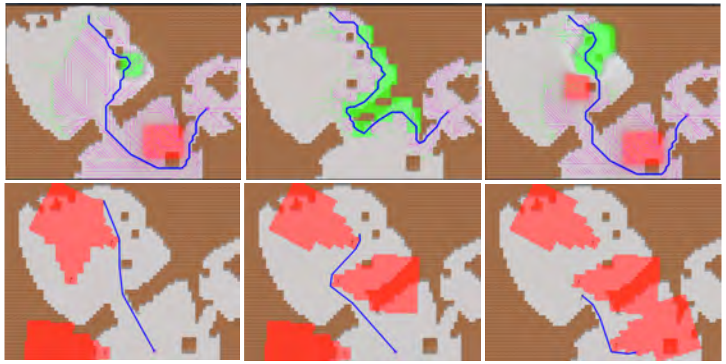
Constraint-Aware Navigation in Dynamic Environments

~~Scaling to large worlds and many agents~~

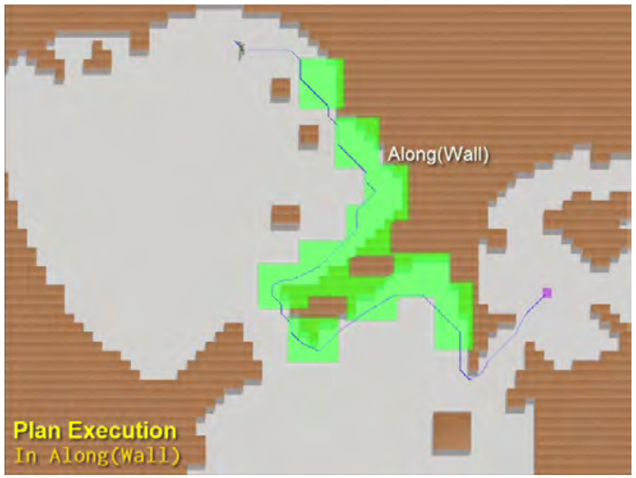
Anytime Dynamic Search on the GPU

www.cs.rutgers.edu/~mubbasir

Global Navigation with Spatial Constraints in Dynamic Environments



www.cs.rutgers.edu/~mubbasir



Extending AD* to compute and repair constraint-aware paths

www.cs.rutgers.edu/~mubbasir

Challenges

Environment representation

Constraint specification

Constraint Satisfaction

www.cs.rutgers.edu/~mubbasir

Proposed Solutions

~~Environment representation~~

Hybrid representation for constraint-aware navigation

Constraint specification

Constraint Satisfaction

www.cs.rutgers.edu/~mubbasir

Proposed Solutions

~~Environment representation~~

Hybrid representation for constraint-aware navigation

~~Constraint specification~~

Cost multiplier fields used to represent qualitative constraints

Constraint Satisfaction

www.cs.rutgers.edu/~mubbasir

Proposed Solutions

~~Environment representation~~

Hybrid representation for constraint-aware navigation

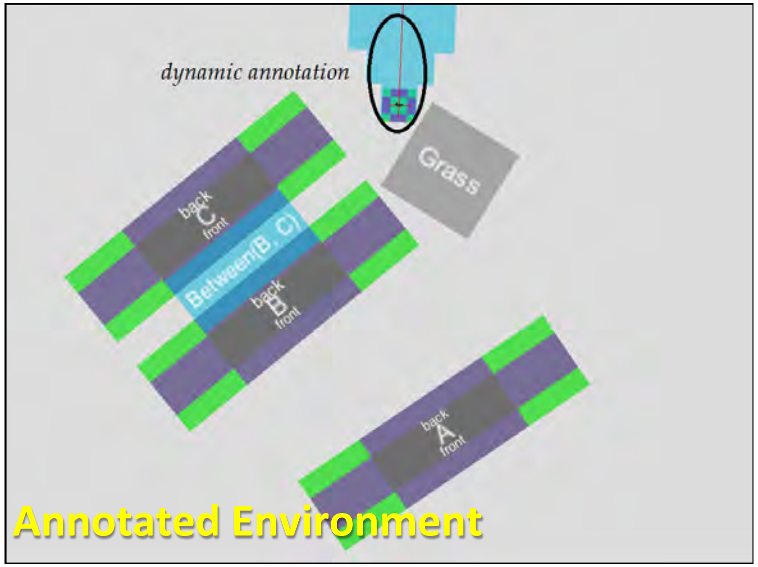
~~Constraint specification~~

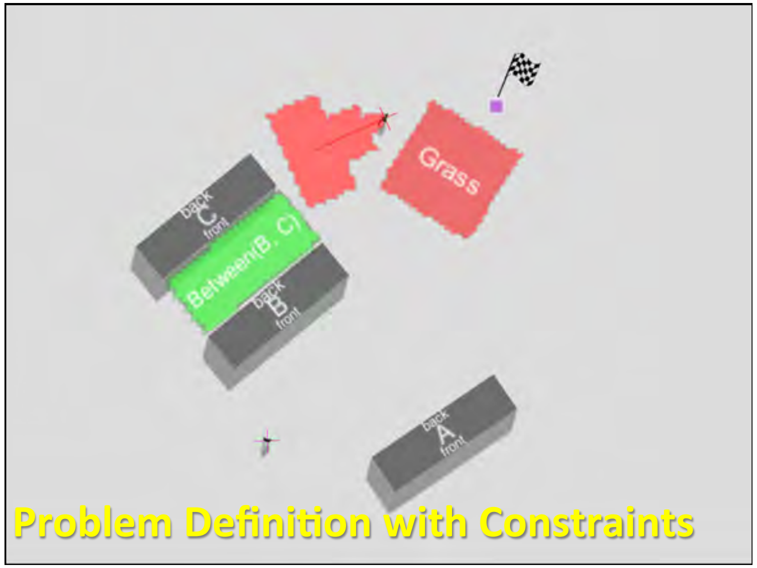
Cost multiplier fields used to represent qualitative constraints

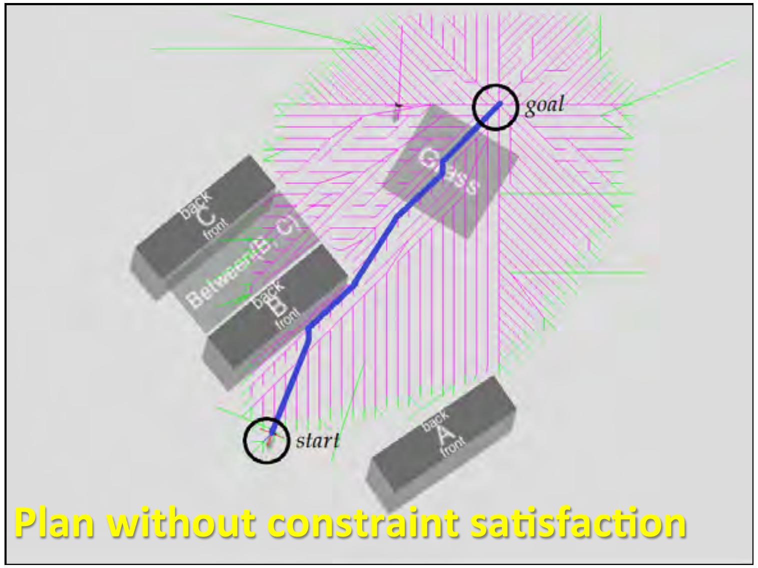
~~Constraint Satisfaction~~

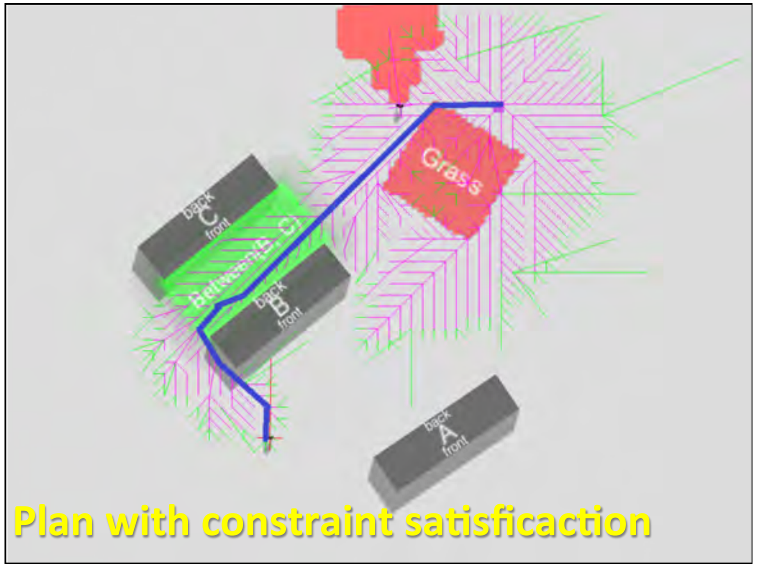
An anytime dynamic planner that computes and repairs constraint-aware paths

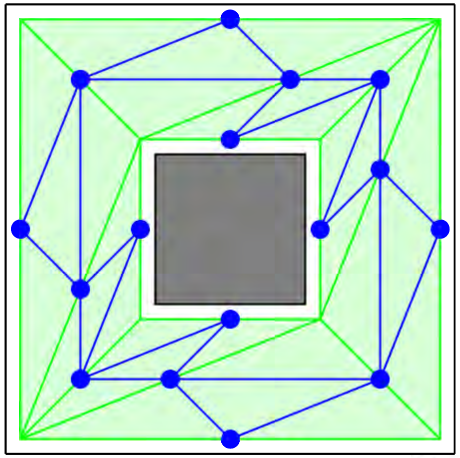
www.cs.rutgers.edu/~mubbasir





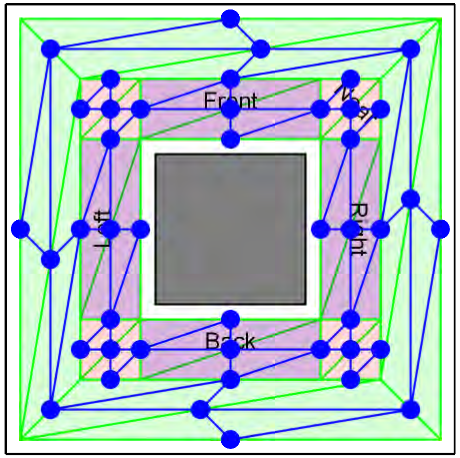






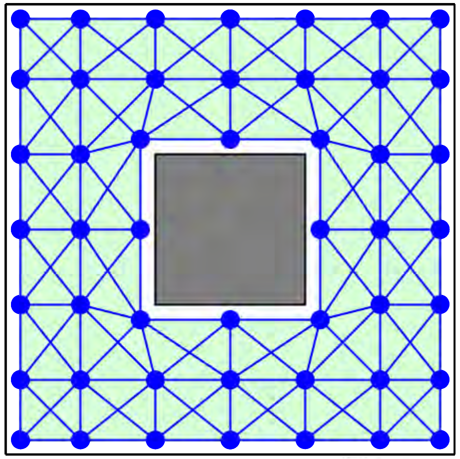
Environment Triangulation Σ_{tri}

www.cs.rutgers.edu/~mubbasir



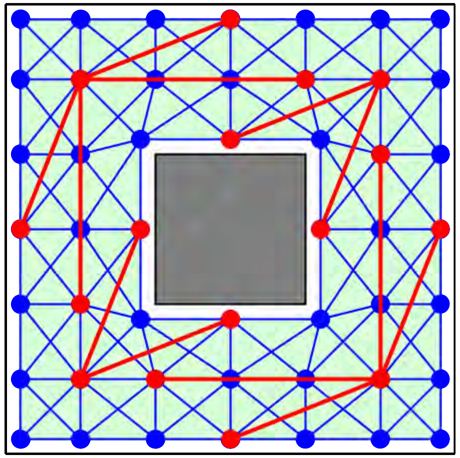
Object annotations with additional nodes added to Σ_{tri}

www.cs.rutgers.edu/~mubbasir



Dense Uniform Graph Σ_{dense}

www.cs.rutgers.edu/~mubbasir



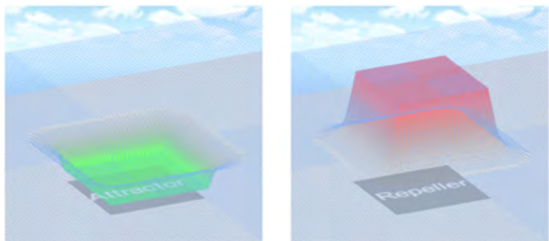
Hybrid graph Σ_{hybrid}

www.cs.rutgers.edu/~mubbasir

Constraint Formulation

$$\overline{m}_c(\vec{x}) = -w\left(k_1 + k_2 \cdot r(c,\vec{x})\right)^{-2}$$

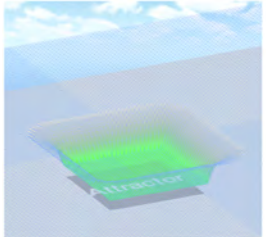
$$m_c(\vec{x}) = \begin{cases} m_c(\vec{x}) & : \; |\overline{m}_c(\vec{x})| < \epsilon \\ 0 & : \; \text{otherwise} \end{cases}$$

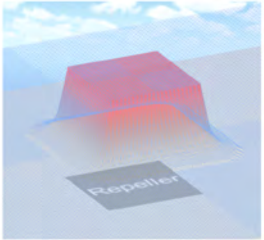


Constraint Formulation

$$\overline{m}_c(\vec{x}) = -w\left(k_1 + k_2 \cdot r(c, \vec{x})\right)^{-2}$$

$$m_c(\vec{x}) = \begin{cases} m_c(\vec{x}) & : \ |\overline{m}_c(\vec{x})| < \epsilon \\ 0 & : \ \text{otherwise} \end{cases}$$

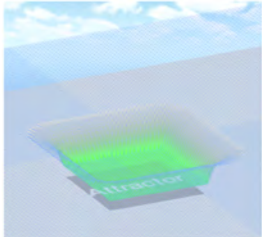


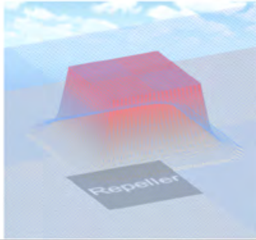


Constraint Formulation

$$\overline{m}_c(\vec{x}) = -w\left(k_1 + k_2 \cdot r(c, \vec{x})\right)^{-2}$$

$$m_c(\vec{x}) = \begin{cases} m_c(\vec{x}) & : \ |\overline{m}_c(\vec{x})| < \epsilon \\ 0 & : \ \text{otherwise} \end{cases}$$

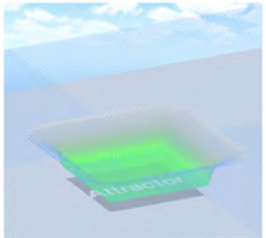


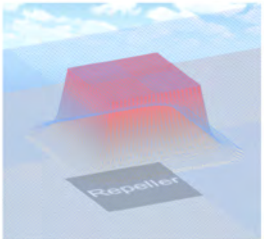


Constraint Formulation

$$\overline{m}_c(\vec{x}) = -w\left(k_1 + k_2 \cdot r(c, \vec{x})\right)^{-2}$$

$$m_c(\vec{x}) = \begin{cases} m_c(\vec{x}) & : \ |\overline{m}_c(\vec{x})| < \epsilon \\ 0 & : \ \text{otherwise} \end{cases}$$

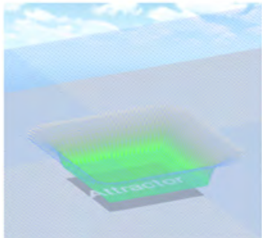


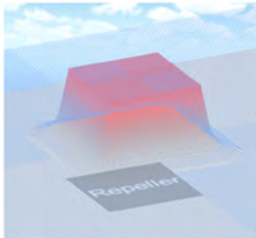


Constraint Formulation

$$\overline{m}_c(\vec{x}) = -w\left(k_1 + k_2 \cdot r(c, \vec{x})\right)^{-2}$$

$$m_c(\vec{x}) = \begin{cases} m_c(\vec{x}) & : \ |\overline{m}_c(\vec{x})| < \epsilon \\ 0 & : \ \text{otherwise} \end{cases}$$





Multiple Constraints

$$m_{\mathbf{C}}\left(\vec{x}\right) = \max\left(1, m_0 + \sum_{c \in \mathbf{C}} m_c\left(\vec{x}\right)\right)$$

Cost multiplier for a transition:

$$M_{\mathbf{C}}\left(s, s'\right) = \int_{s \to s'} m_{\mathbf{C}}\left(\vec{x}\right) d\vec{x}$$

$$M_{\mathbf{C}}\left(s, s'\right) \approx m_{\mathbf{C}}\left(\frac{\vec{x}_s + \vec{x}_{s'}}{2}\right)$$

www.cs.rutgers.edu/~mubbasir

Multiple Constraints

$$m_{\mathbf{C}}\left(\vec{x}\right) = \max\left(1, m_0 + \sum_{c \in \mathbf{C}} m_c\left(\vec{x}\right)\right)$$

Cost multiplier for a transition:

$$M_{\mathbf{C}}\left(s, s'\right) = \int_{s \to s'} m_{\mathbf{C}}\left(\vec{x}\right) d\vec{x}$$

$$M_{\mathbf{C}}\left(s, s'\right) \approx m_{\mathbf{C}}\left(\frac{\vec{x}_s + \vec{x}_{s'}}{2}\right)$$

www.cs.rutgers.edu/~mubbasir

Multiple Constraints

$$m_{\mathbf{C}}\left(\vec{x}\right) = \max\left(1, m_0 + \sum_{c \in \mathbf{C}} m_c\left(\vec{x}\right)\right)$$

Cost multiplier for a transition:

$$M_{\mathbf{C}}\left(s, s'\right) = \int_{s \to s'} m_{\mathbf{C}}\left(\vec{x}\right) d\vec{x}$$

$$M_{\mathbf{C}}\left(s, s'\right) \approx m_{\mathbf{C}}\left(\frac{\vec{x}_s + \vec{x}_{s'}}{2}\right)$$

www.cs.rutgers.edu/~mubbasir

Multiple Constraints

$$m_{\mathbf{C}}\left(\vec{x}\right) = \max\left(1, m_0 + \sum_{c \in \mathbf{C}} m_c\left(\vec{x}\right)\right)$$

Cost multiplier for a transition:

$$M_{\mathbf{C}}\left(s, s'\right) = \int_{s \to s'} m_{\mathbf{C}}\left(\vec{x}\right) d\vec{x}$$

$$M_{\mathbf{C}}\left(s, s'\right) \approx m_{\mathbf{C}}\left(\frac{\vec{x}_s + \vec{x}_{s'}}{2}\right)$$

www.cs.rutgers.edu/~mubbasir

Multiple Constraints

$$m_{\mathbf{C}}(\vec{x}) = \max\left(1, m_0 + \sum_{c \in \mathbf{C}} m_c(\vec{x})\right)$$

Cost multiplier for a transition:

$$M_{\mathbf{C}}(s, s') = \int_{s \to s'} m_{\mathbf{C}}(\vec{x})\, d\vec{x}$$

$$M_{\mathbf{C}}(s, s') \approx m_{\mathbf{C}}\left(\frac{\vec{x}_s + \vec{x}_{s'}}{2}\right)$$

www.cs.rutgers.edu/~mubbasir

Planner: Cost Computation

Modified cost of reaching state s:

$$g(s_{\text{start}}, s) = g(s_{\text{start}}, s') + M_{\mathbf{C}}(s, s') \cdot c(s, s')$$

$$g(s_{\text{start}}, s) = \sum_{(s_i, s_j) \in \Pi(s_{\text{start}}, s)} M_{\mathbf{C}}(s_i, s_j) \cdot c(s_i, s_j)$$

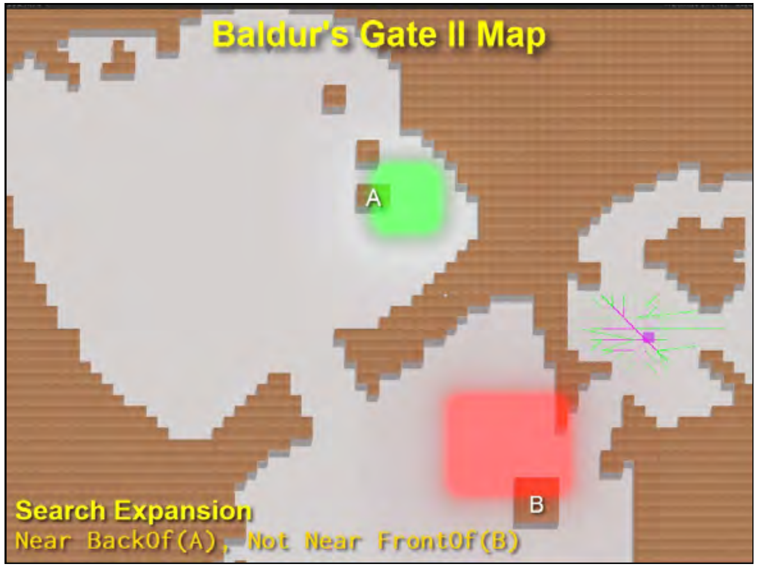
www.cs.rutgers.edu/~mubbasir

Accommodating Dynamic Constraints

Algorithm 1 ConstraintChangeUpdate $(c, \vec{x}_{\text{prev}}, \vec{x}_{\text{next}})$

```
S_c^prev = region(m_c, x_prev)
S_c^next = region(m_c, x_next)
for each s ∈ S_c^prev ∪ S_c^next do
    if pred(s) ⋂ VISITED ≠ NULL then
        UpdateState(s)
    if s' ∈ S_c^next ∧ c ∈ C_h then g(s') = ∞
        if s' ∈ CLOSED then
            for each s'' ∈ succ(s') do
                if s'' ∈ VISITED then
                    UpdateState(s'')
```

www.cs.rutgers.edu/~mubbasir



Proposed Solutions

~~Real-time Planning in Dynamic Environments~~

From Classical A* to Anytime Dynamic Search

~~Planning with Constraints~~

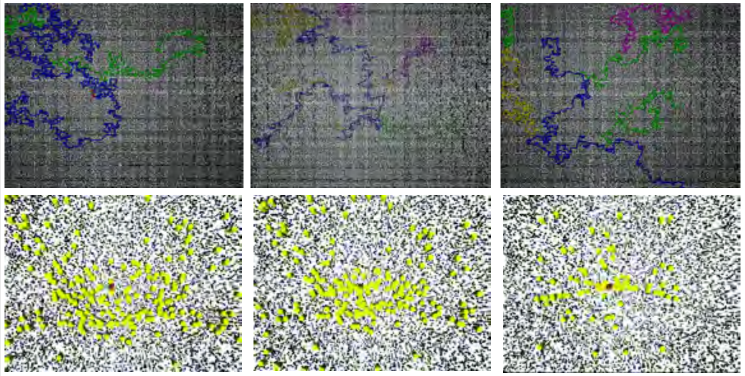
Constraint-Aware Navigation in Dynamic Environments

~~Scaling to large worlds and many agents~~

Anytime Dynamic Search on the GPU

www.cs.rutgers.edu/~mubbasir

Need for high fidelity navigation in complex, dynamic virtual environments



www.cs.rutgers.edu/~mubbasir

Challenges

Large-scale, complex, dynamic environments

Strict optimality requirements

Scalability with number of agents

Proposed Solutions

~~Large-scale, complex, dynamic environments~~

Massively parallel wave-front based search with efficient plan repair

Strict optimality requirements

Scalability with number of agents

www.cs.rutgers.edu/~mubbasir

Proposed Solutions

~~Large-scale, complex, dynamic environments~~

Massively parallel wave-front based search with efficient plan repair

~~Strict optimality requirements~~

Termination condition enforces strict optimality with minimum number of GPU iterations

Scalability with number of agents

www.cs.rutgers.edu/~mubbasir

Proposed Solutions

~~Large-scale, complex, dynamic environments~~

Massively parallel wave-front based search with efficient plan repair

~~Strict optimality requirements~~

Termination condition enforces strict optimality with minimum number of GPU iterations

~~Scalability with number of agents~~

Handles any number of moving agents at no additional computational cost

www.cs.rutgers.edu/~mubbasir

Method Overview

Algorithm 1 ***computePlan***$(*m_{cpu})$

$m_r \leftarrow m_{cpu}$
$m_w \leftarrow m_{cpu}$
repeat
  $flag \leftarrow 0$
  plannerKernel$(m_r, m_w, flag)$
  swap (m_r, m_w)
until $(flag = 0)$
$m_{cpu} \leftarrow m_r$

www.cs.rutgers.edu/~mubbasir

Method Overview

Algorithm 1 ***computePlan***$(*m_{cpu})$

$m_r \leftarrow m_{cpu}$
$m_w \leftarrow m_{cpu}$
repeat
  $flag \leftarrow 0$
  plannerKernel$(m_r, m_w, flag)$
  swap (m_r, m_w)
until $(flag = 0)$
$m_{cpu} \leftarrow m_r$

www.cs.rutgers.edu/~mubbasir

Method Overview

Algorithm 1 ***computePlan***$(*m_{cpu})$

$m_r \leftarrow m_{cpu}$
$m_w \leftarrow m_{cpu}$
repeat
  $flag \leftarrow 0$
  plannerKernel$(m_r, m_w, flag)$
  swap (m_r, m_w)
until $(flag = 0)$
$m_{cpu} \leftarrow m_r$

www.cs.rutgers.edu/~mubbasir

Method Overview

Algorithm 2 ***plannerKernel***$(*m_r, *m_w, *flag)$

$s \leftarrow threadState$
if $s \neq obstacle \wedge s \neq goal$ **then**
  for all s' in $neighbor(s)$ **do**
    if $s' \neq obstacle$ **then**
      $newg \leftarrow g(s') + c(s, s')$
      if $(newg < g(s) \vee g(s) = -1) \wedge g(s') > -1$ **then**
        $pred(s) \leftarrow s'$
        $g(s) \leftarrow newg$
        { evaluate_termination_condition }

$$g(s) = min_{s' \in succ(s) \wedge g(s') \geq 0}(c(s, s') + g(s'))$$

www.cs.rutgers.edu/~mubbasir

Method Overview

Algorithm 2 ***plannerKernel***$(*m_r, *m_w, *flag)$

$s \leftarrow threadState$
if $s \neq obstacle \wedge s \neq goal$ **then**
  for all s' in $neighbor(s)$ **do**
    if $s' \neq obstacle$ **then**
      $newg \leftarrow g(s') + c(s, s')$
      if $(newg < g(s) \vee g(s) = -1) \wedge g(s') > -1$ **then**
        $pred(s) \leftarrow s'$
        $g(s) \leftarrow newg$
        { evaluate_termination_condition }

$$g(s) = min_{s' \in succ(s) \wedge g(s') \geq 0}(c(s, s') + g(s'))$$

Method Overview

Algorithm 2 ***plannerKernel***$(*m_r, *m_w, *flag)$

$s \leftarrow threadState$
if $s \neq obstacle \wedge s \neq goal$ **then**
  for all s' in $neighbor(s)$ **do**
    if $s' \neq obstacle$ **then**
      $newg \leftarrow g(s') + c(s, s')$
      if $(newg < g(s) \vee g(s) = -1) \wedge g(s') > -1$ **then**
        $pred(s) \leftarrow s'$
        $g(s) \leftarrow newg$
        { evaluate_termination_condition }

$$g(s) = min_{s' \in succ(s) \wedge g(s') \geq 0}(c(s, s') + g(s'))$$

Method Overview

Algorithm 1 ***computePlan***$(*m_{cpu})$

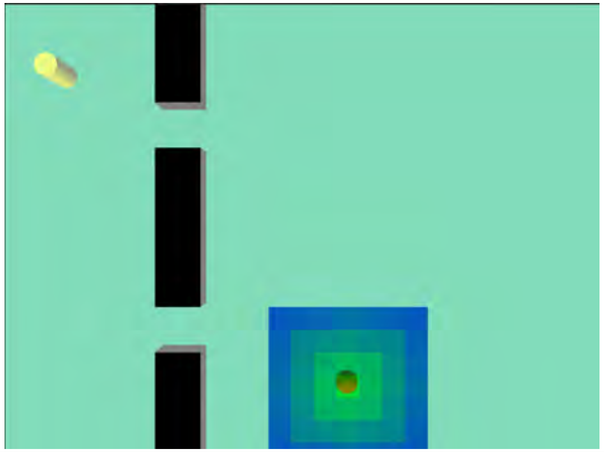
$m_r \leftarrow m_{cpu}$
$m_w \leftarrow m_{cpu}$
repeat
  $flag \leftarrow 0$
  plannerKernel$(m_r, m_w, flag)$
  swap (m_r, m_w)
until $(flag = 0)$
$m_{cpu} \leftarrow m_r$

Method Overview

Algorithm 1 ***computePlan***$(*m_{cpu})$

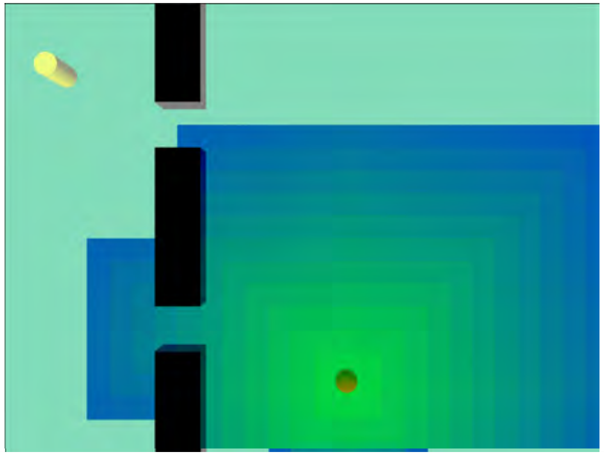
$m_r \leftarrow m_{cpu}$
$m_w \leftarrow m_{cpu}$
repeat
  $flag \leftarrow 0$
  plannerKernel$(m_r, m_w, flag)$
  swap (m_r, m_w)
until $(flag = 0)$
$m_{cpu} \leftarrow m_r$

Wavefront expansion. N =3



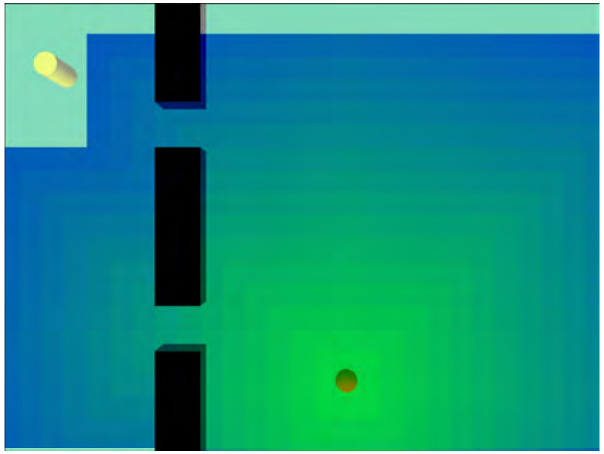
www.cs.rutgers.edu/~mubbasir

Wavefront expansion. N =11



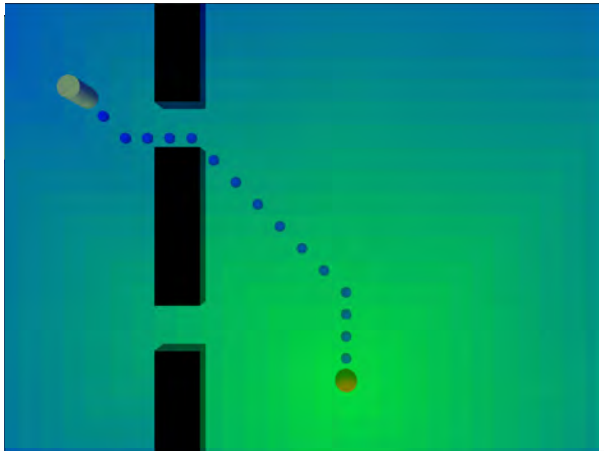
www.cs.rutgers.edu/~mubbasir

Wavefront expansion. N =15



www.cs.rutgers.edu/~mubbasir

Wavefront expansion. N =18



www.cs.rutgers.edu/~mubbasir

Termination Conditions

Exit when goal reached

$$\textbf{if}(s == goal)\texttt{flag} = 0$$

Exit when whole map converges

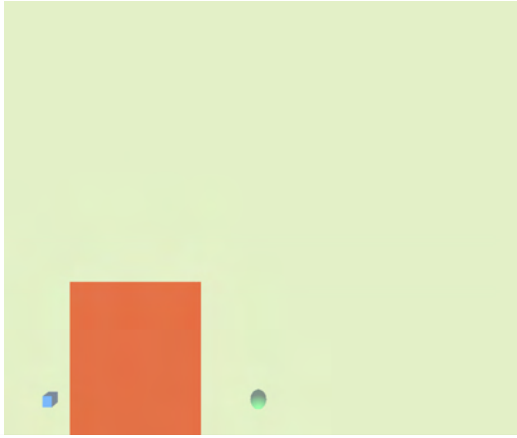
$$\texttt{flag} = 1$$

Minimal map convergence with optimality guarantees

$$\textbf{if}(g(s) < g(start) \vee g(agent) = -1) flag = 1$$

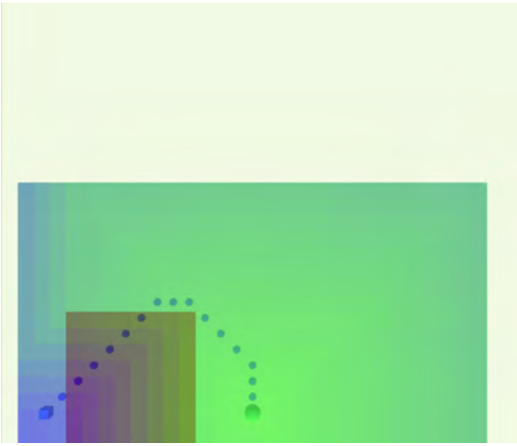
www.cs.rutgers.edu/~mubbasir

Non-uniform state space



www.cs.rutgers.edu/~mubbasir

Sub-optimal solution, N = 8



www.cs.rutgers.edu/~mubbasir

Termination Conditions

Exit when goal reached

$$\textbf{if}(s == goal)\texttt{flag} = 0$$

Exit when whole map converges

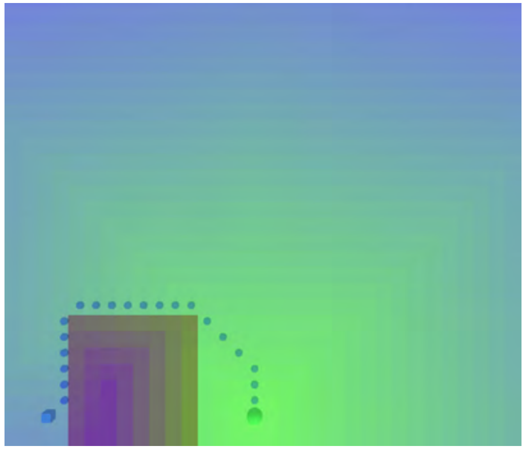
$$\texttt{flag} = 1$$

Minimal map convergence with optimality guarantees

$$\textbf{if}(g(s) < g(start) \vee g(agent) = -1) flag = 1$$

www.cs.rutgers.edu/~mubbasir

Optimal solution, N = 17



Termination Conditions

Exit when goal reached

$$\textbf{if}(s == goal)\texttt{flag} = 0$$

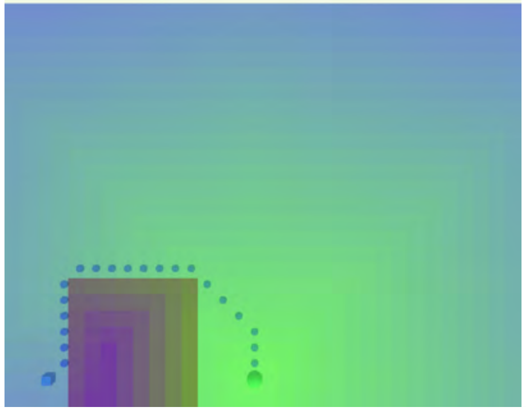
Exit when whole map converges

$$\texttt{flag} = 1$$

- **Minimal map convergence with optimality guarantees**

$$\textbf{if}(g(s) < g(start) \vee g(agent) = -1) flag = 1$$

Optimal solution, N = 12 (minimum number of iterations)



Efficient Plan Repair for Dynamic Environments & Moving Agents

Algorithm 3 Algorithm to propagate state inconsistency

$s \leftarrow threadState$
if $pred(s) \neq NULL$ **then**
  if $(g(s) == obstacle \vee pred(s) == obstacle \vee g(s) \neq g(pred(s)) + c(s, s'))$ **then**
    $pred(s) = NULL$
    $g(s) = -1$
    $incons = \texttt{true}$

Multi-Agent Planning

Extended Termination Condition

$$\textbf{if}((g(s) < max_{a_i \in \{a\}} g(a_i)) \vee (g(a_i) = -1 \forall a_i \in \{a\}))$$

Multi-Agent Simulation

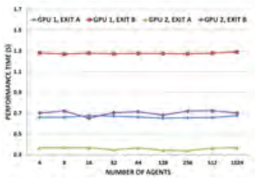
- Single map can be queried by all agents to compute path
- Movement along path using local collision avoidance

Multiple Target Locations

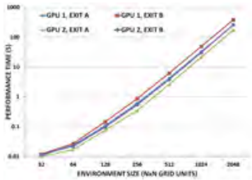
- A separate map required for each target
- Significant memory overhead

www.cs.rutgers.edu/~mubbasir

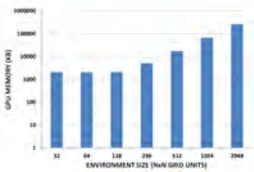
Performance Analysis



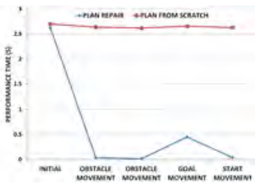
Agent Scalability



Environment Scalability

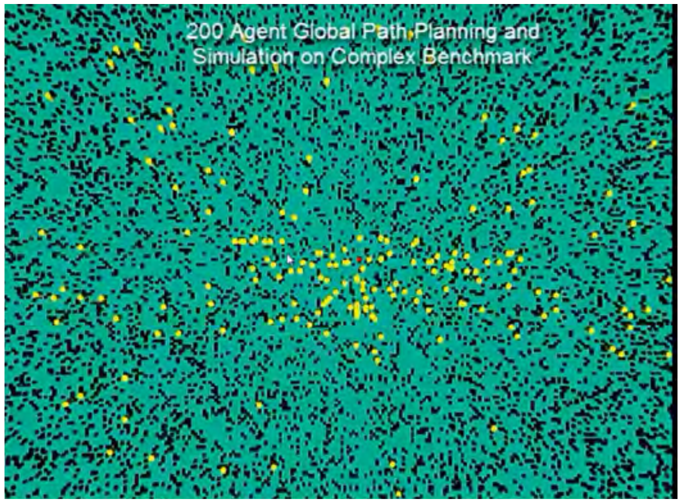


Memory

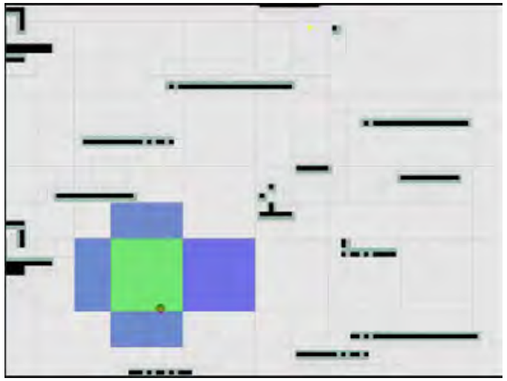


Dynamic Planning

www.cs.rutgers.edu/~mubbasir



GPU-based Dynamic Search on Adaptive Resolution Grids



GPU-based Dynamic Search on Adaptive Resolution Grids
Francisco Garcia, Mubbasir Kapadia, and Norman I. Badler
IEEE International Conference on Robotics and Automation, June 2014

www.cs.rutgers.edu/~mubbasir

Render the Possibilities
SIGGRAPH2016

Planning Techniques for Character Animation

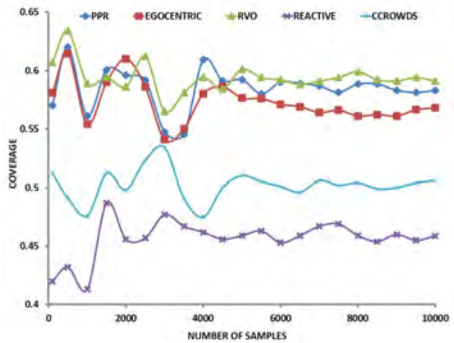
Mubbasir Kapadia

www.cs.rutgers.edu/~mubbasir

Outline

- Footstep Domain for Dynamic Crowds
- Precomputing Environment Semantics for Contact-Rich Character Animation
- Additional Application Domains

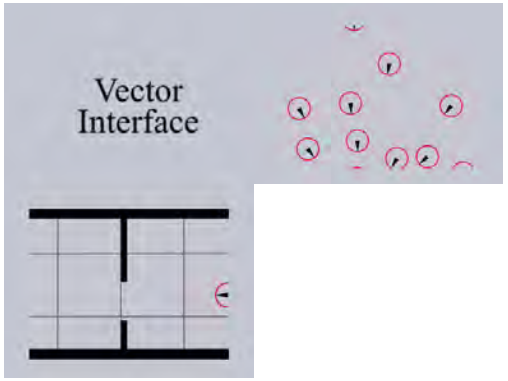
"Current steering algorithms cannot handle the space of all possible scenarios that agents encounter in dynamic virtual environments"



Scenario Space: Characterizing Coverage, Quality, and Failure of Steering Algorithms
Mubbasir Kapadia, Matt Wang, Shawn Singh, Glenn Reinmann, Petros Faloutsos
ACM SIGGRAPH Symposium on Computer Animation, August 2011

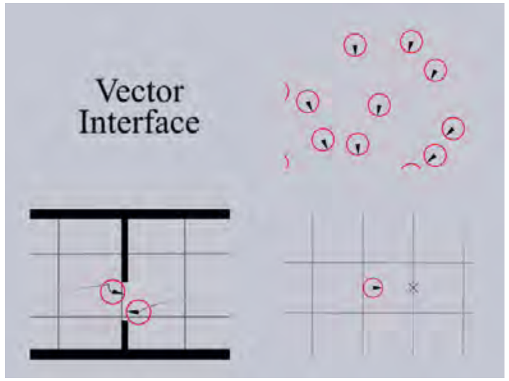
"A particle-based agent representation cannot capture nuanced interactions that humans exhibit in confined and crowded situations."

"A particle-based agent representation cannot capture nuanced interactions that humans exhibit in confined and crowded situations."



www.cs.rutgers.edu/~mubbasir

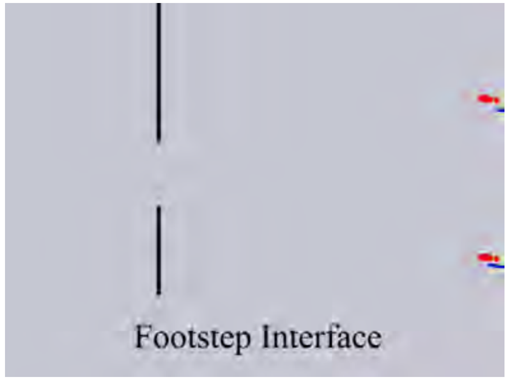
"A particle-based agent representation cannot capture nuanced interactions that humans exhibit in confined and crowded situations."



www.cs.rutgers.edu/~mubbasir

www.cs.rutgers.edu/~mubbasir

Footstep Navigation for Dynamic Crowds



Footstep Navigation for Dynamic Crowds
Shawn Singh, Mubbasir Kapadia, Glenn Reinmann, Petros Faloutsos
Special Issue, Computer Animation and Virtual Worlds (CAVW), April 2011

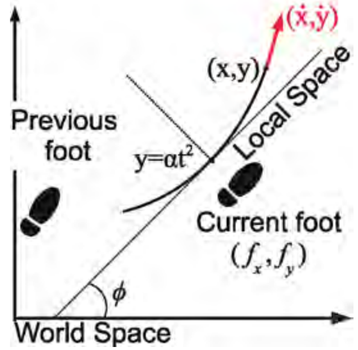
State and Action Space

$$\mathbf{S} = \{s | \mathbf{x}, \mathbf{v}, \mathbf{x}_f, \theta_f, I \in \{L, R\}\}$$
$$\mathbf{A} = \{a | \phi, v_{des}, T\}$$

Footstep Domain

www.cs.rutgers.edu/~mubbasir

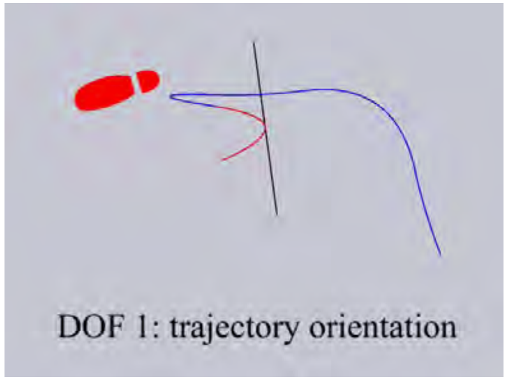
Footstep Action Selection



$$(x(t), y(t), \dot{x}(t), \dot{y}(t)) = (v_{x_0}t, \alpha t^2, v_{x_0}, 2\alpha t)$$

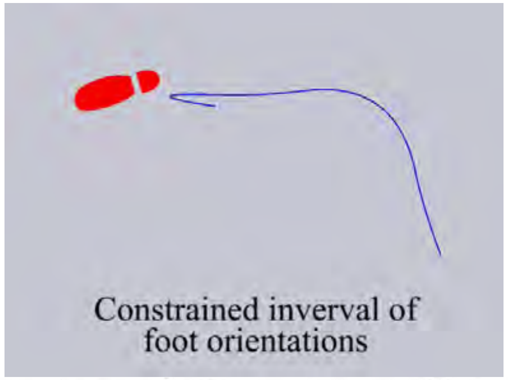
www.cs.rutgers.edu/~mubbasir

Footstep Action Selection



www.cs.rutgers.edu/~mubbasir

Footstep Orientation



$$[f_{\phi\text{next\_inner}}, f_{\phi\text{next\_outer}}] = \left[f_{\phi\text{prev\_outer}}, f_{\phi\text{prev\_inner}} + \frac{\pi}{2}\right] \cap [\phi, \text{atan2}(\dot{y}, \dot{x})]$$

www.cs.rutgers.edu/~mubbasir

Cost Formulation

Cost Function

$$c(s, s') = \Delta E_1 + \Delta E_2 + \Delta E_3$$

$$\Delta E_1 = R \cdot T$$

$$\Delta E_2 = \frac{m}{2} \left| (v_{\text{desired}})^2 - (v_0 \cos(2\theta))^2 \right|$$

$$\Delta E_3 = w \cdot \frac{dP}{dt} \cdot \text{length} = w \cdot m\alpha \cdot \text{length}$$

Heuristic Function

$$h(s) = c_{\text{expected}} \times n$$

Cost Formulation

Cost Function

$$c(s, s') = \Delta E_1 + \Delta E_2 + \Delta E_3$$

$$\Delta E_1 = R \cdot T$$

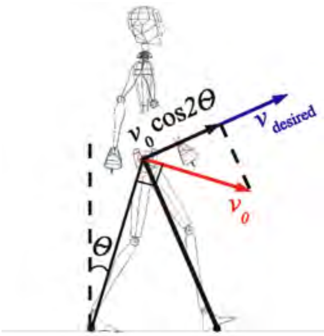
$$\Delta E_2 = \frac{m}{2} \left| (v_{\text{desired}})^2 - (v_0 \cos(2\theta))^2 \right|$$

$$\Delta E_3 = w \cdot \frac{dP}{dt} \cdot \text{length} = w \cdot m\alpha \cdot \text{length}$$

Heuristic Function

$$h(s) = c_{\text{expected}} \times n$$

Spherical Inverted Pendulum Model – Sagittal View



$$\Delta E_2 = \frac{m}{2} \left| (v_{\text{desired}})^2 - (v_0 \cos(2\theta))^2 \right|$$

Cost Formulation

Cost Function

$$c(s, s') = \Delta E_1 + \Delta E_2 + \Delta E_3$$

$$\Delta E_1 = R \cdot T$$

$$\Delta E_2 = \frac{m}{2} \left| (v_{\text{desired}})^2 - (v_0 \cos(2\theta))^2 \right|$$

$$\Delta E_3 = w \cdot \frac{dP}{dt} \cdot \text{length} = w \cdot m\alpha \cdot \text{length}$$

Heuristic Function

$$h(s) = c_{\text{expected}} \times n$$

Cost Formulation

Cost Function

$$c(s,s') = \Delta E_1 + \Delta E_2 + \Delta E_3$$

$$\Delta E_1 = R \cdot T$$

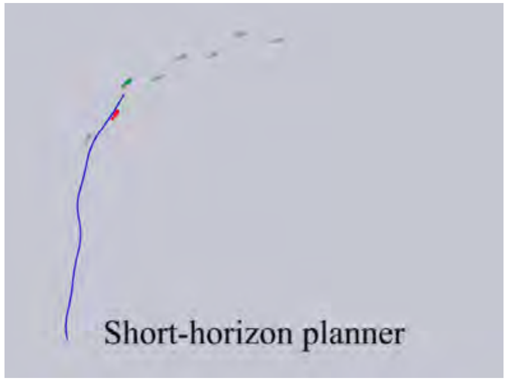
$$\Delta E_2 = \frac{m}{2}\left|(v_{\text{desired}})^2 - (v_0 \cos(2\theta))^2\right|$$

$$\Delta E_3 = w \cdot \frac{dP}{dt} \cdot \text{length} = w \cdot m\alpha \cdot \text{length}$$

Heuristic Function

$$h(s) = c_{\text{expected}} \times n$$

Short Horizon Space-Time Planner



www.cs.rutgers.edu/~mubbasir

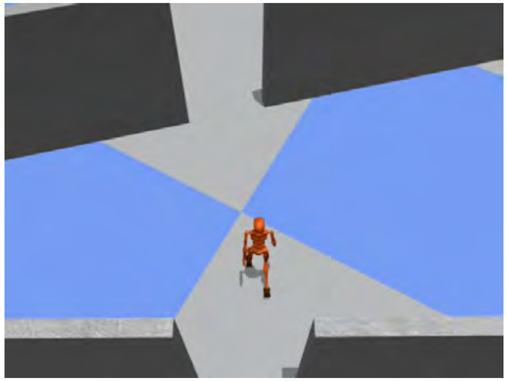
U Turn

www.cs.rutgers.edu/~mubbasir

Dynamic Collision Bounds

www.cs.rutgers.edu/~mubbasir

Narrow Passageways

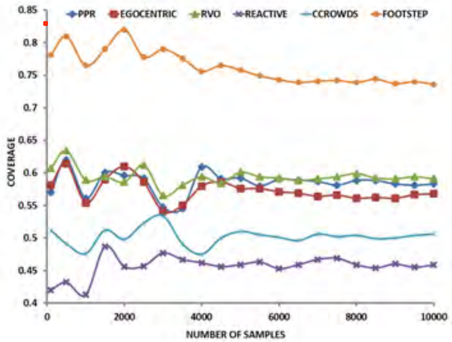


www.cs.rutgers.edu/~mubbasir

Large Scale Simulations

www.cs.rutgers.edu/~mubbasir

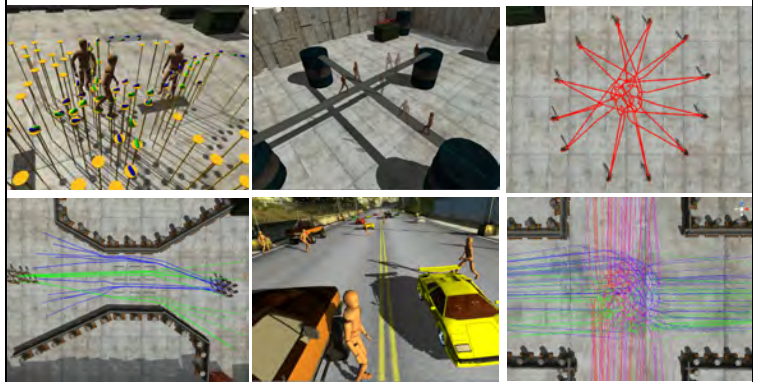
Coverage Comparison



Scenario Space: Characterizing Coverage, Quality, and Failure of Steering Algorithms
Mubbasir Kapadia, Matt Wang, Shawn Singh, Glenn Reinmann, Petros Faloutsos
ACM SIGGRAPH Symposium on Computer Animation, August 2011

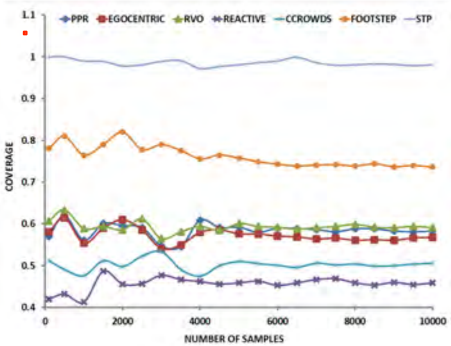
www.cs.rutgers.edu/~mubbasir

Need for high fidelity navigation in complex, dynamic virtual environments



www.cs.rutgers.edu/~mubbasir

The Quest for Complete Coverage



Scenario Space: Characterizing Coverage, Quality, and Failure of Steering Algorithms
Mubbasir Kapadia, Matt Wang, Shawn Singh, Glenn Reinmann, Petros Faloutsos
ACM SIGGRAPH Symposium on Computer Animation, August 2011

www.cs.rutgers.edu/~mubbasir

Outline

- Footstep Domain for Dynamic Crowds
- **Precomputing Environment Semantics for Contact-Rich Character Animation**
- Additional Application Domains

www.cs.rutgers.edu/~mubbasir

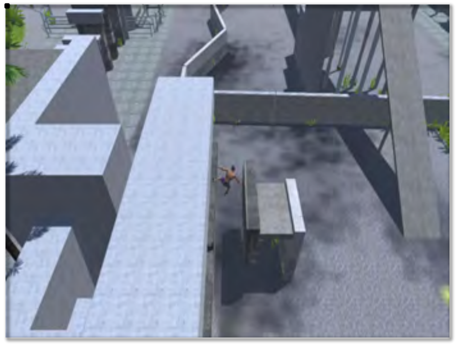
PRECISION: Precomputed Environment Semantics for Contact-Rich Character Animation
Mubbasir Kapadia, Xu Xianghao, Maurizio Nitti, Marcelo Kallmann,
Stelian Coros, Robert W. Sumner, Markus Gross
ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D), 2016

REQUIREMENTS

- Scalability in environment and motion complexity
- Interactivity
- Coupled character and environment authoring

REQUIREMENTS

Scalability in environment and motion complexity

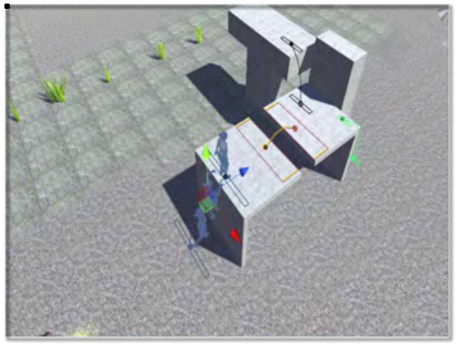


REQUIREMENTS

Interactivity

REQUIREMENTS

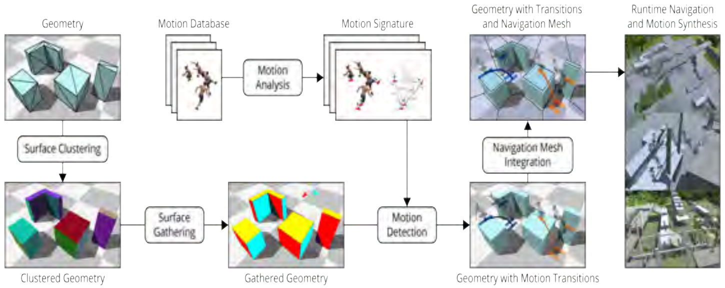
Coupled character and environment authoring



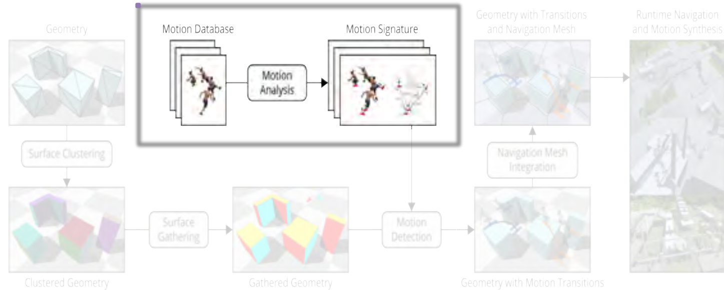
SOLUTIONS

- **Motion Analysis**: Identify contact semantics
- **Environment Analysis:** Identify how characters can interact with geometry
- **Runtime Navigation & Motion Synthesis:** Seamless integration with existing approaches

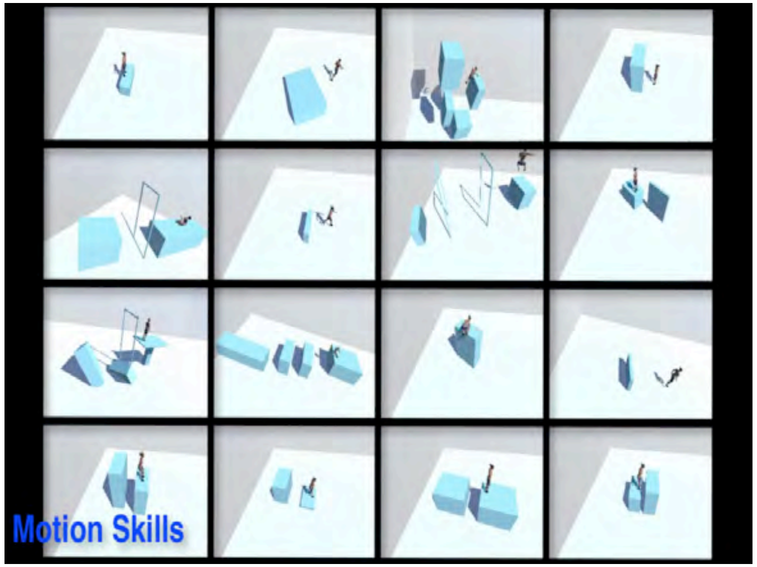
PRECISION Framework



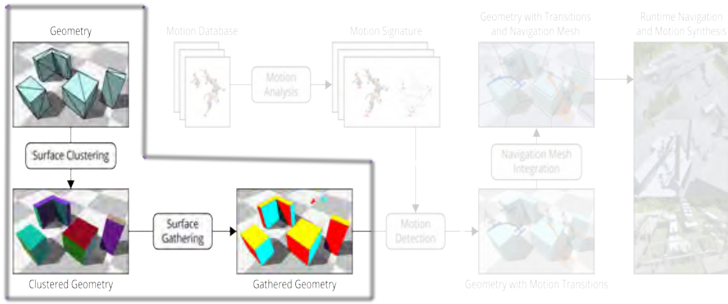
PRECISION Framework



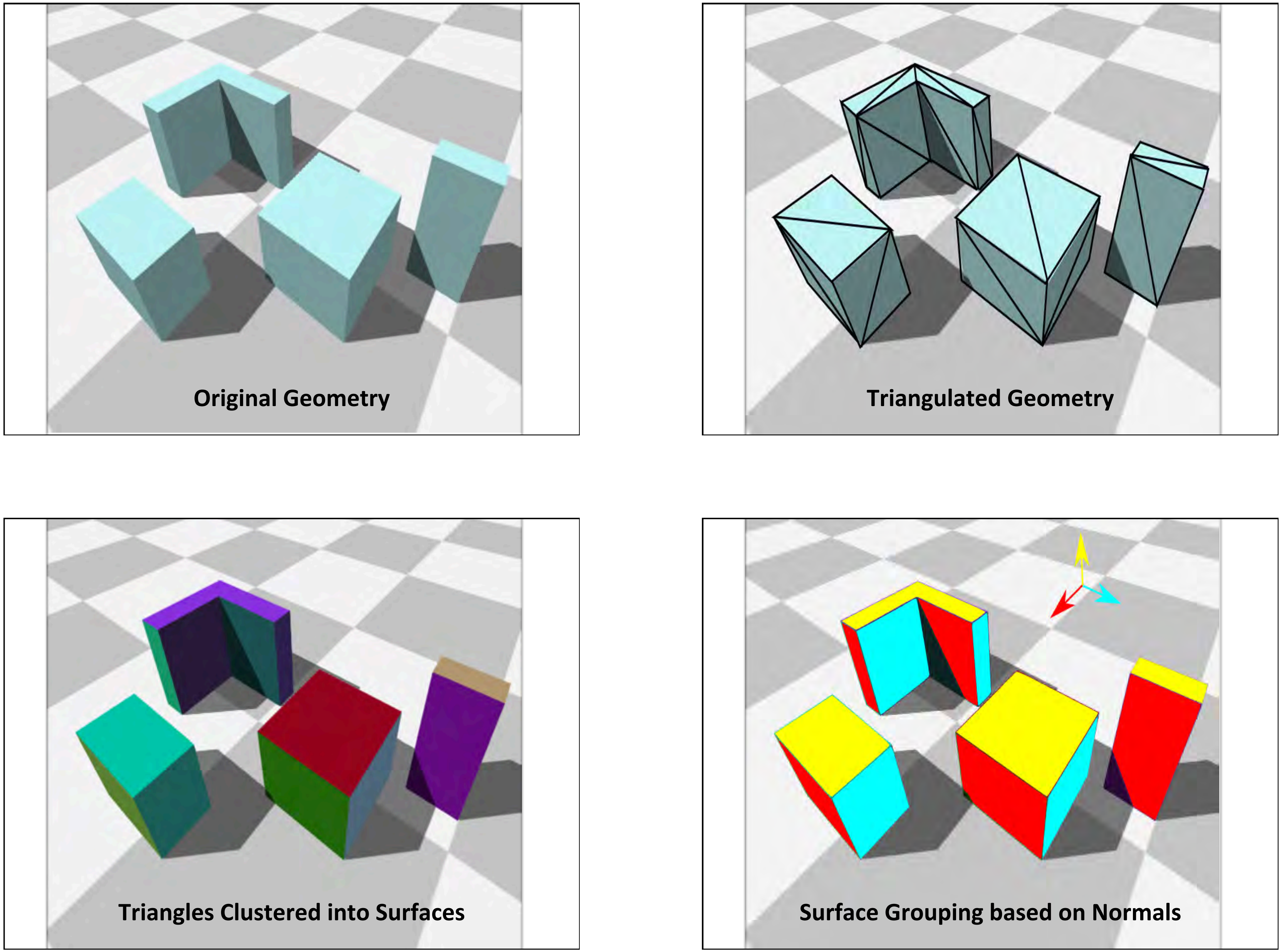
Motion Analysis



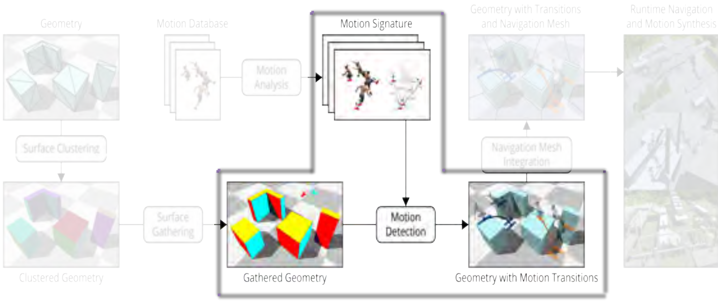
PRECISION Framework



Environment Analysis

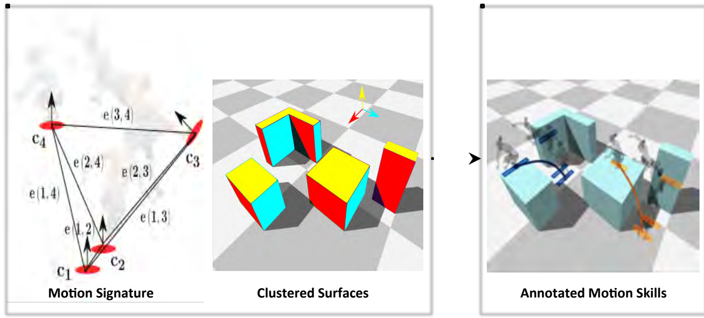

Original Geometry
Triangulated Geometry
Triangles Clustered into Surfaces
Surface Grouping based on Normals

PRECISION Framework



Motion Skill Detection

Motion Skill Detection



Algorithm 2: DetectMotions (M, S)

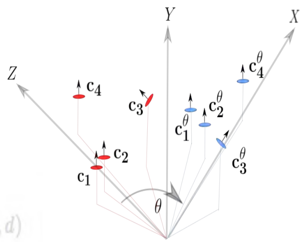
foreach $m \in M$ **do**
- **foreach** $\theta \in (0, 2\pi)$ **do**
  - **foreach** $i \in (1, |C_m|)$ **do**
    - $c_i^{\theta} = \text{Rotate}(c_i, \theta)$
    - **foreach** $s \in S$ **do**
      - **if** $n_s \cdot n_{c_i^{\theta}} \leq \epsilon$ **then**
        - $S_i \leftarrow S_i \cup s$
  - $R \leftarrow S_1$
  - **foreach** $i \in (2, |C_m|)$ **do**
    - $R_t \leftarrow \emptyset$
    - **foreach** $(s_a, s_b) \in R \times S_i$ **do**
      - $\hat{v}_{i-1,i} \leftarrow \frac{p_{i-1} - p_i}{||p_{i-1} - p_i||}$
      - $\langle s_a', s_b' \rangle \leftarrow \text{ProjectIntersect}(s_a, s_b, \hat{v}_{i-1,i}, d)$
      - $R_t \leftarrow R_t \cup \langle s_a', s_b' \rangle$
    - $R \leftarrow R_t$
  - $R \leftarrow \text{CollisionCheck}(R, V_m)$;
  - **return** R

Algorithm 2: DetectMotions (M, S)

foreach $m \in M$ **do**
- **foreach** $\theta \in (0, 2\pi)$ **do**
  - **foreach** $i \in (1, |C_m|)$ **do**
    - $c_i^{\theta} = \text{Rotate}(c_i, \theta)$
    - **foreach** $s \in S$ **do**
      - **if** $n_s \cdot n_{c_i^{\theta}} \leq \epsilon$ **then**
        - $S_i \leftarrow S_i \cup s$
  - $R \leftarrow S_1$
  - **foreach** $i \in (2, |C_m|)$ **do**
    - $R_t \leftarrow \emptyset$
    - **foreach** $(s_a, s_b) \in R \times S_i$ **do**
      - $\hat{v}_{i-1,i} \leftarrow \frac{p_{i-1} - p_i}{||p_{i-1} - p_i||}$
      - $\langle s_a', s_b' \rangle \leftarrow \text{ProjectIntersect}(s_a, s_b, \hat{v}_{i-1,i}, d)$
      - $R_t \leftarrow R_t \cup \langle s_a', s_b' \rangle$
    - $R \leftarrow R_t$
  - $R \leftarrow \text{CollisionCheck}(R, V_m)$;
  - **return** R

Algorithm 2: DetectMotions (M, S)

```
foreach m ∈ M do
    foreach θ ∈ (0, 2π) do
        foreach i ∈ (1, |C_m|) do
            c_i^θ = Rotate(c_i, θ)
            foreach s ∈ S do
                if n_s · n_{c_i^θ} ≤ ε then
                    S_i ← S_i ∪ s
        R ← S_1
        foreach i ∈ (2, |C_m|) do
            R_t ← ∅
            foreach (s_a, s_b) ∈ R × S_i do
                v̂_{i−1,i} ← (p_{i−1} − p_i) / ||p_{i−1} − p_i||
                ⟨s'_a, s'_b⟩ ← ProjectIntersect(s_a, s_b, v̂_{i−1,i}, d)
                R_t ← R_t ∪ ⟨s'_a, s'_b⟩
            R ← R_t
        R ← CollisionCheck(R, V_m);
        return R
```



Contact Rotation

Algorithm 2: DetectMotions (M, S)

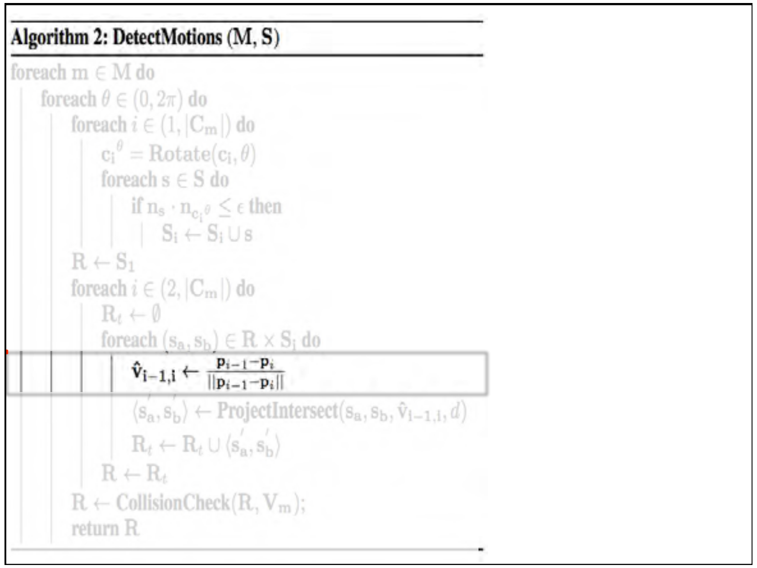
```
foreach m ∈ M do
    foreach θ ∈ (0, 2π) do
        foreach i ∈ (1, |C_m|) do
            c_i^θ = Rotate(c_i, θ)
            foreach s ∈ S do
                if n_s · n_{c_i^θ} ≤ ε then
                    S_i ← S_i ∪ s
        R ← S_1
        foreach i ∈ (2, |C_m|) do
            R_t ← ∅
            foreach (s_a, s_b) ∈ R × S_i do
                v̂_{i−1,i} ← (p_{i−1} − p_i) / ||p_{i−1} − p_i||
                ⟨s'_a, s'_b⟩ ← ProjectIntersect(s_a, s_b, v̂_{i−1,i}, d)
                R_t ← R_t ∪ ⟨s'_a, s'_b⟩
            R ← R_t
        R ← CollisionCheck(R, V_m);
        return R
```

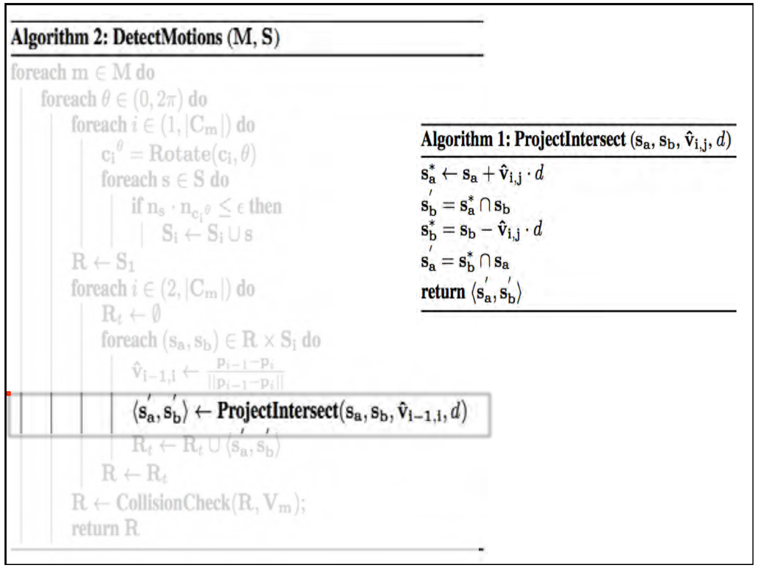
Algorithm 2: DetectMotions (M, S)

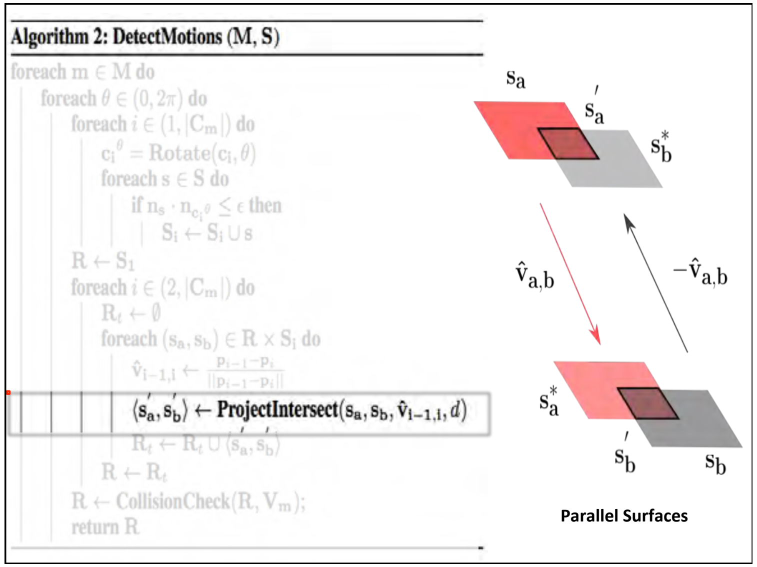
```
foreach m ∈ M do
    foreach θ ∈ (0, 2π) do
        foreach i ∈ (1, |C_m|) do
            c_i^θ = Rotate(c_i, θ)
            foreach s ∈ S do
                if n_s · n_{c_i^θ} ≤ ε then
                    S_i ← S_i ∪ s
        R ← S_1
        foreach i ∈ (2, |C_m|) do
            R_t ← ∅
            foreach (s_a, s_b) ∈ R × S_i do
                v̂_{i−1,i} ← (p_{i−1} − p_i) / ||p_{i−1} − p_i||
                ⟨s'_a, s'_b⟩ ← ProjectIntersect(s_a, s_b, v̂_{i−1,i}, d)
                R_t ← R_t ∪ ⟨s'_a, s'_b⟩
            R ← R_t
        R ← CollisionCheck(R, V_m);
        return R
```

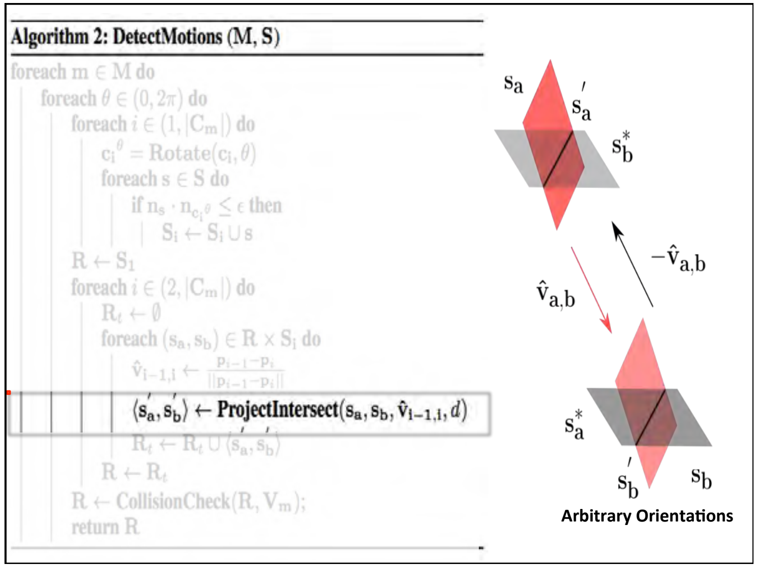
Algorithm 2: DetectMotions (M, S)

```
foreach m ∈ M do
    foreach θ ∈ (0, 2π) do
        foreach i ∈ (1, |C_m|) do
            c_i^θ = Rotate(c_i, θ)
            foreach s ∈ S do
                if n_s · n_{c_i^θ} ≤ ε then
                    S_i ← S_i ∪ s
        R ← S_1
        foreach i ∈ (2, |C_m|) do
            R_t ← ∅
            foreach (s_a, s_b) ∈ R × S_i do
                v̂_{i−1,i} ← (p_{i−1} − p_i) / ||p_{i−1} − p_i||
                ⟨s'_a, s'_b⟩ ← ProjectIntersect(s_a, s_b, v̂_{i−1,i}, d)
                R_t ← R_t ∪ ⟨s'_a, s'_b⟩
            R ← R_t
        R ← CollisionCheck(R, V_m);
        return R
```


Algorithm 2: DetectMotions (M, S)
foreach m ∈ M do
foreach θ ∈ (0, 2π) do
foreach i ∈ (1, |C_m|) do
c_i^θ = Rotate(c_i, θ)
foreach s ∈ S do
if n_s · n_{c_i^θ} ≤ ε then
S_i ← S_i ∪ s
R ← S_1
foreach i ∈ (2, |C_m|) do
R_t ← ∅
foreach (s_a, s_b) ∈ R × S_i do
v̂_{i−1,i} ← (p_{i−1} − p_i) / ||p_{i−1} − p_i||
⟨s'_a, s'_b⟩ ← ProjectIntersect(s_a, s_b, v̂_{i−1,i}, d)
R_t ← R_t ∪ ⟨s'_a, s'_b⟩
R ← R_t
R ← CollisionCheck(R, V_m);
return R


Algorithm 2: DetectMotions (M, S)
foreach m ∈ M do
foreach θ ∈ (0, 2π) do
foreach i ∈ (1, |C_m|) do
c_i^θ = Rotate(c_i, θ)
foreach s ∈ S do
if n_s · n_{c_i^θ} ≤ ε then
S_i ← S_i ∪ s
R ← S_1
foreach i ∈ (2, |C_m|) do
R_t ← ∅
foreach (s_a, s_b) ∈ R × S_i do
v̂_{i−1,i} ← (p_{i−1} − p_i) / ||p_{i−1} − p_i||
⟨s'_a, s'_b⟩ ← ProjectIntersect(s_a, s_b, v̂_{i−1,i}, d)
R_t ← R_t ∪ ⟨s'_a, s'_b⟩
R ← R_t
R ← CollisionCheck(R, V_m);
return R
Algorithm 1: ProjectIntersect (s_a, s_b, v̂_{i,j}, d)
s*_a ← s_a + v̂_{i,j} · d
s'_b = s*_a ∩ s_b
s*_b = s_b − v̂_{i,j} · d
s'_a = s*_b ∩ s_a
return ⟨s'_a, s'_b⟩


Algorithm 2: DetectMotions (M, S)
foreach m ∈ M do
foreach θ ∈ (0, 2π) do
foreach i ∈ (1, |C_m|) do
c_i^θ = Rotate(c_i, θ)
foreach s ∈ S do
if n_s · n_{c_i^θ} ≤ ε then
S_i ← S_i ∪ s
R ← S_1
foreach i ∈ (2, |C_m|) do
R_t ← ∅
foreach (s_a, s_b) ∈ R × S_i do
v̂_{i−1,i} ← (p_{i−1} − p_i) / ||p_{i−1} − p_i||
⟨s'_a, s'_b⟩ ← ProjectIntersect(s_a, s_b, v̂_{i−1,i}, d)
R_t ← R_t ∪ ⟨s'_a, s'_b⟩
R ← R_t
R ← CollisionCheck(R, V_m);
return R
s_a
s'_a
s*_b
v̂_{a,b}
−v̂_{a,b}
s*_a
s'_b
s_b
Parallel Surfaces


Algorithm 2: DetectMotions (M, S)
foreach m ∈ M do
foreach θ ∈ (0, 2π) do
foreach i ∈ (1, |C_m|) do
c_i^θ = Rotate(c_i, θ)
foreach s ∈ S do
if n_s · n_{c_i^θ} ≤ ε then
S_i ← S_i ∪ s
R ← S_1
foreach i ∈ (2, |C_m|) do
R_t ← ∅
foreach (s_a, s_b) ∈ R × S_i do
v̂_{i−1,i} ← (p_{i−1} − p_i) / ||p_{i−1} − p_i||
⟨s'_a, s'_b⟩ ← ProjectIntersect(s_a, s_b, v̂_{i−1,i}, d)
R_t ← R_t ∪ ⟨s'_a, s'_b⟩
R ← R_t
R ← CollisionCheck(R, V_m);
return R
s_a
s'_a
s*_b
−v̂_{a,b}
v̂_{a,b}
s*_a
s'_b
s_b
Arbitrary Orientations

Algorithm 2: DetectMotions (M, S)

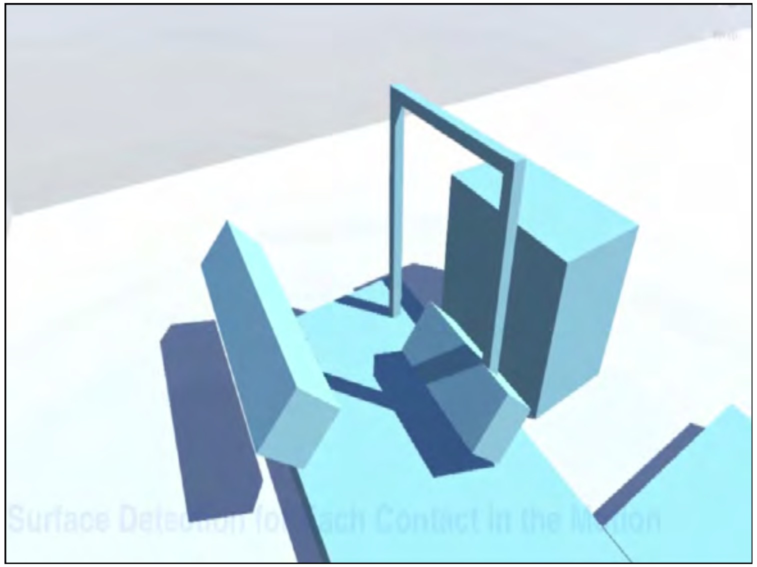
foreach $m \in M$ do
  foreach $\theta \in (0, 2\pi)$ do
    foreach $i \in (1, |C_m|)$ do
      $c_i^{\theta} = \text{Rotate}(c_i, \theta)$
      foreach $s \in S$ do
        if $n_s \cdot n_{c_i^{\theta}} \leq \epsilon$ then
          $S_i \leftarrow S_i \cup s$
    $R \leftarrow S_1$
    foreach $i \in (2, |C_m|)$ do
      $R_t \leftarrow \emptyset$
      foreach $(s_a, s_b) \in R \times S_i$ do
        $\hat{v}_{i-1,i} \leftarrow \frac{p_{i-1} - p_i}{||p_{i-1} - p_i||}$
        $\langle s_a', s_b' \rangle \leftarrow \text{ProjectIntersect}(s_a, s_b, \hat{v}_{i-1,i}, d)$
        $\mathbf{R_t \leftarrow R_t \cup \langle s_a', s_b' \rangle}$
      $R \leftarrow R_t$
    $R \leftarrow \text{CollisionCheck}(R, V_m)$;
    return R

Algorithm 2: DetectMotions (M, S)

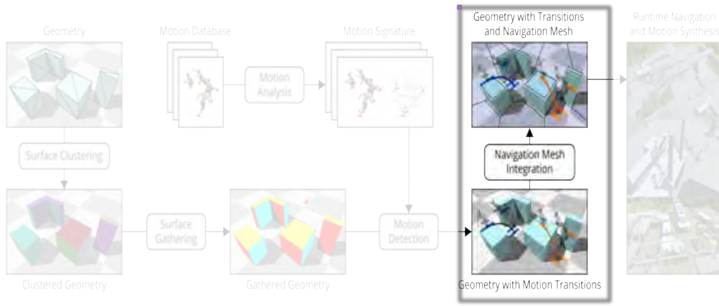
foreach $m \in M$ do
  foreach $\theta \in (0, 2\pi)$ do
    foreach $i \in (1, |C_m|)$ do
      $c_i^{\theta} = \text{Rotate}(c_i, \theta)$
      foreach $s \in S$ do
        if $n_s \cdot n_{c_i^{\theta}} \leq \epsilon$ then
          $S_i \leftarrow S_i \cup s$
    $R \leftarrow S_1$
    foreach $i \in (2, |C_m|)$ do
      $R_t \leftarrow \emptyset$
      foreach $(s_a, s_b) \in R \times S_i$ do
        $\hat{v}_{i-1,i} \leftarrow \frac{p_{i-1} - p_i}{||p_{i-1} - p_i||}$
        $\langle s_a', s_b' \rangle \leftarrow \text{ProjectIntersect}(s_a, s_b, \hat{v}_{i-1,i}, d)$
        $R_t \leftarrow R_t \cup \langle s_a', s_b' \rangle$
      $R \leftarrow R_t$
    $\mathbf{R \leftarrow \text{CollisionCheck}(R, V_m)}$;
    return R

Collision-free motion

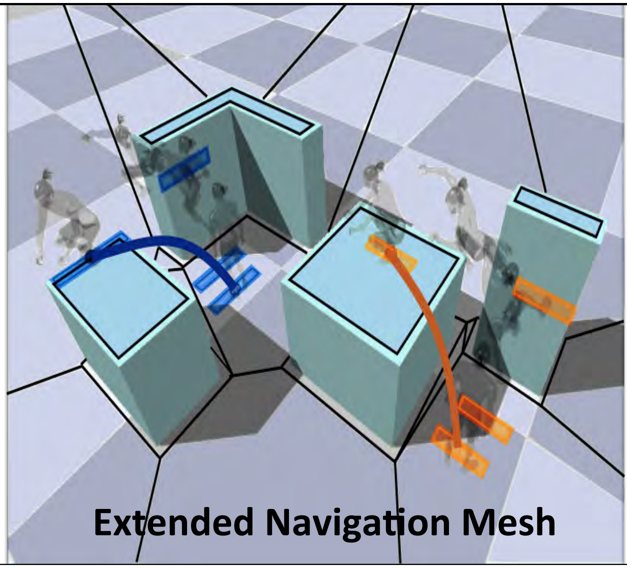
Motion with collision

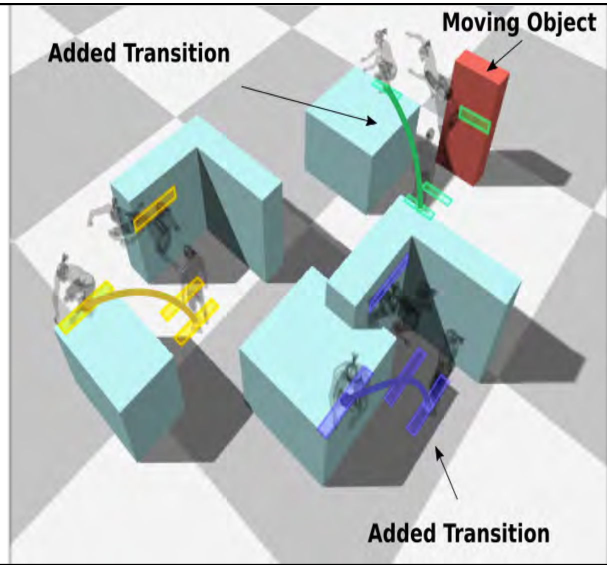


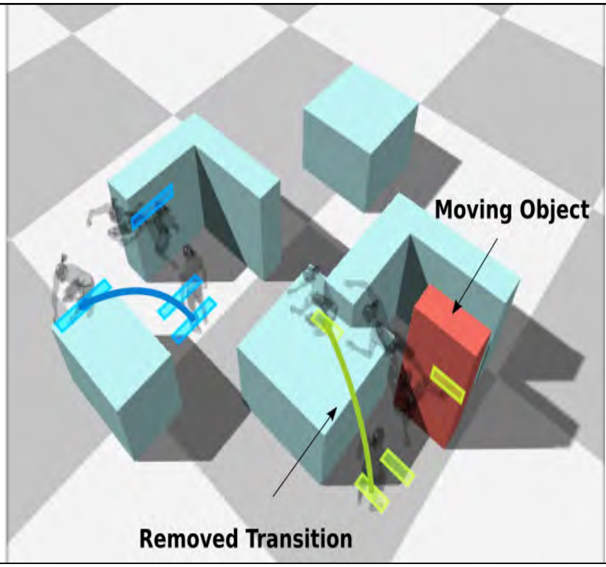
PRECISION Framework



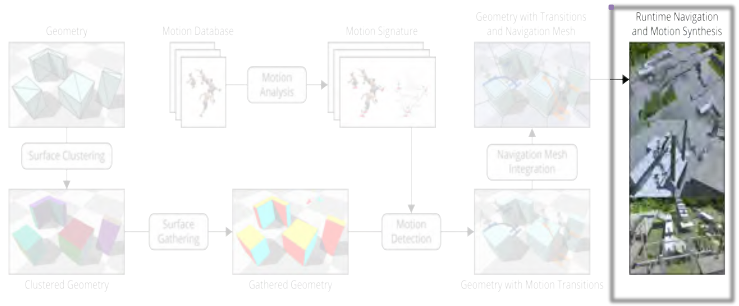
Navigation Integration



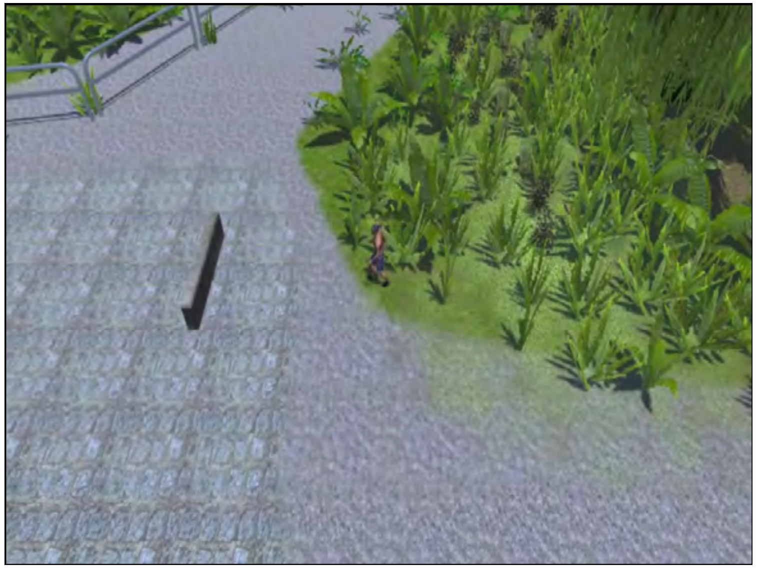




PRECISION Framework



Runtime Navigation and Motion Synthesis



Dynamic Game Worlds

Outline

- Footstep Domain for Dynamic Crowds
- Precomputing Environment Semantics for Contact-Rich Character Animation
- **Additional Application Domains**

www.cs.rutgers.edu/~mubbasir

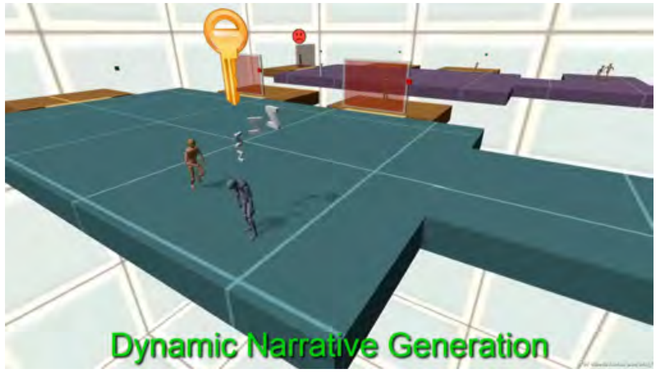
ACCLMesh: Curvature-Based Navigation Mesh Generation

ACCLMesh: Curvature-Based Navigation Mesh Generation
Glen Berseth, Mubbasir Kapadia, Petros Faloutsos
Computer Animation and Virtual Worlds (2016, To Appear)

Multi-Domain Planning for Real-time Planning in Dynamic Environments

Multi-Domain Planning for Real-time Planning in Dynamic Environments
Mubbasir Kapadia, Alejandro Beacco Porres, Francisco Garcia, Nuria Pelechano, Norman. I. Badler
ACM SIGGRAPH/EG Symposium on Computer Animation, 2013

An Event-Centric Planning Approach for Dynamic Real-Time Narrative



An Event-Centric Planning Approach for Dynamic Real-Time Narrative
Alexander Shoulson, Max Gilbert, Mubbasir Kapadia, and Norman I. Badler
International Conference on Motion in Games, 2013

Conclusion

- Planning not limited to simple navigation problems or non-interactive applications.
- Challenges
  - Discretizing problem representation
  - Defining problem domain (state, action space, costs, heuristics)
  - Choosing right planning strategy

www.cs.rutgers.edu/~mubbasir

Render the Possibilities
SIGGRAPH2016

Module III – Planning Techniques for Character Animation

Marcelo Kallmann
mkallmann@ucmerced.edu

UCMERCED
UNIVERSITY OF CALIFORNIA, MERCED

- When to use full-body motion planners?
  - To achieve automatic motion synthesis for virtual characters among obstacles
    - 3D collision detection always needed (bottleneck)
  - Planners can be integrated on top of motion controllers
    - Leveraging the quality for several powerful approaches developed in computer animation
      - Ex.: Motion Control session yesterday

Planning in High Dimensions

- Build graph representation of free space by sampling valid poses/configurations
  - Example graph/roadmap built by sampling:

Planning Collision-Free Reaching Motions for Interactive Object Manipulation and Grasping, Eurographics, 2003

Planning in High Dimensions

Planning Collision-Free Reaching Motions for Interactive Object Manipulation and Grasping, Eurographics, 2003

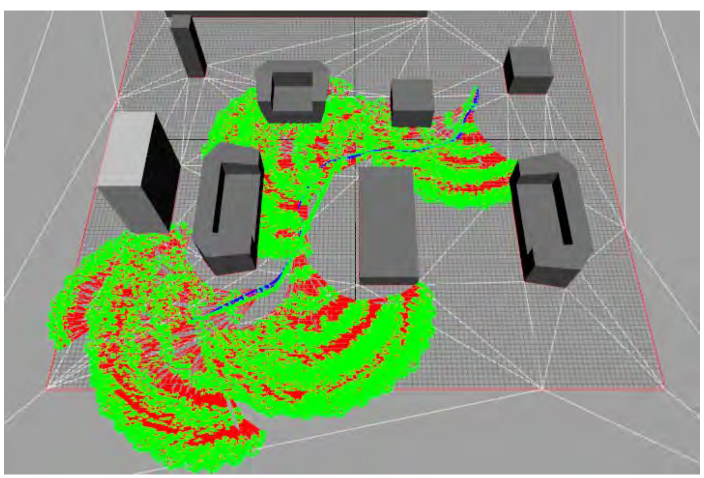
Planning locomotion with motion capture data

Adding Motion Capture Data

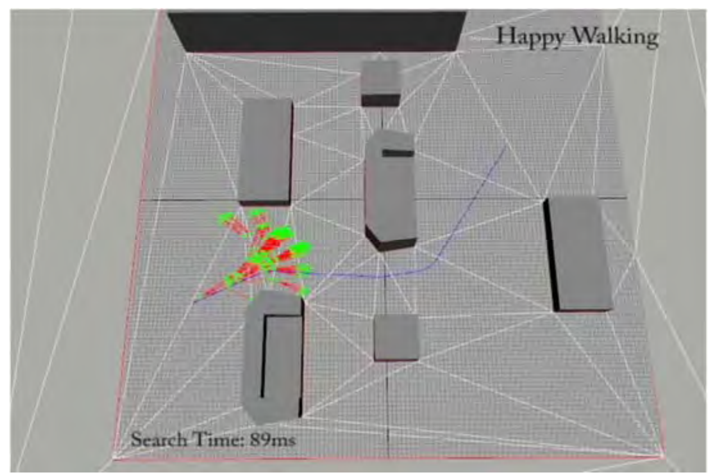
- Example approach
  - Build a motion graph from motion capture data
    - Search on the motion graph (*graph unrolling*)
    - Good quality, but often slow to use directly
  - Possible to improve speed with
    - search precomputation
    - and 2D path planning
- Extensive literature available on the area
  - Representative references in course notes

Speeding up motion search

- By pre-computing search trees per node:

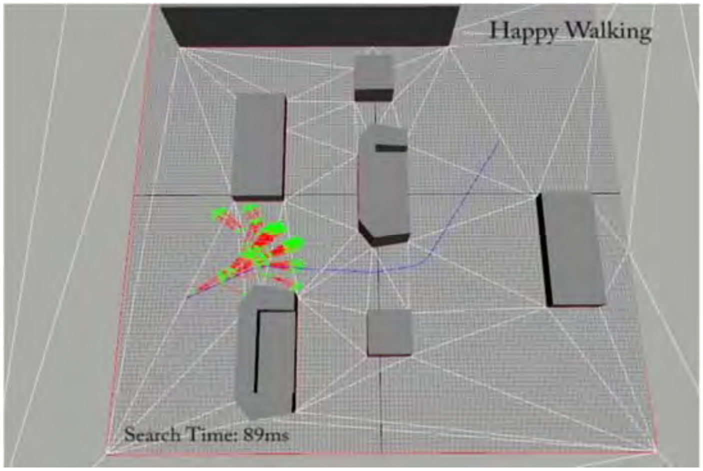


Precomputed Motion Maps



Analyzing Locomotion Synthesis with Feature-Based Motion Graphs, IEEE TVCG 2012
Precomputed Motion Maps for Unstructured Motion Capture, SCA 2012
Feature-Based Locomotion with Inverse Branch Kinematics, best paper at MIG 2011

Precomputed Motion Maps

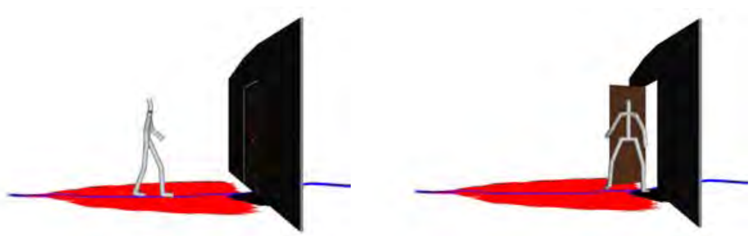


Analyzing Locomotion Synthesis with Feature-Based Motion Graphs, IEEE TVCG 2012
Precomputed Motion Maps for Unstructured Motion Capture, SCA 2012
Feature-Based Locomotion with Inverse Branch Kinematics, best paper at MIG 2011

Integrating manipulation planning with locomotion

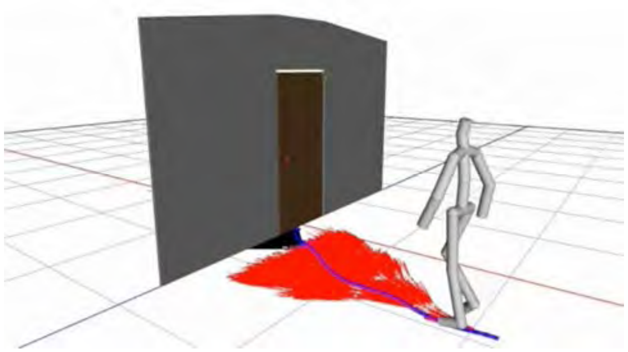
Addressing Full-Body Manipulations

- Integration of two planners
  - Motion capture concatenation search for locomotion
  - Sampling-based planning for the arm



Multi-Modal Data-Driven Motion Planning and Synthesis
Mentar Mahmudi and Marcelo Kallmann
ACM SIGGRAPH Conference on Motion in Games (MIG), 2015

Addressing Full-Body Manipulations

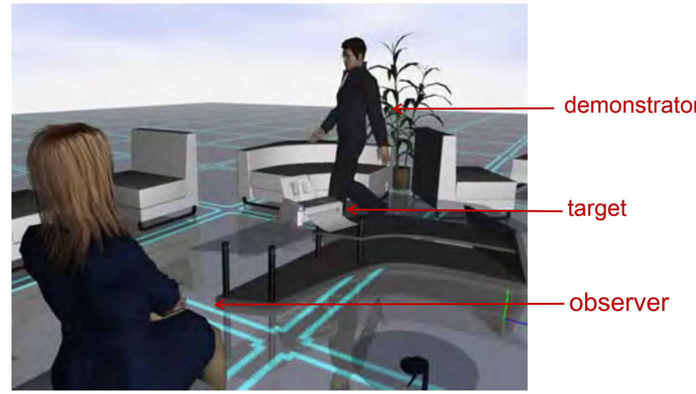


Multi-Modal Data-Driven Motion Planning and Synthesis
Mentar Mahmudi and Marcelo Kallmann
ACM SIGGRAPH Conference on Motion in Games (MIG), 2015

Addressing application-specific coordination constraints

Ex. Application: Virtual Demonstrators

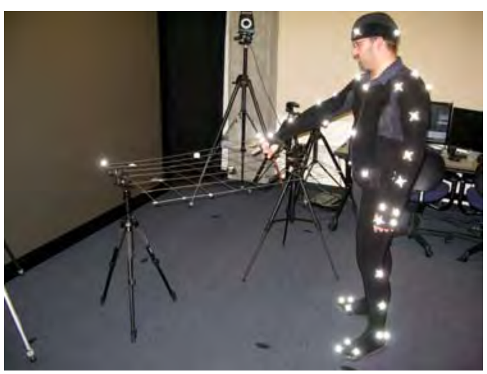
- Determine suitable locations for delivering information, and then animate a solution



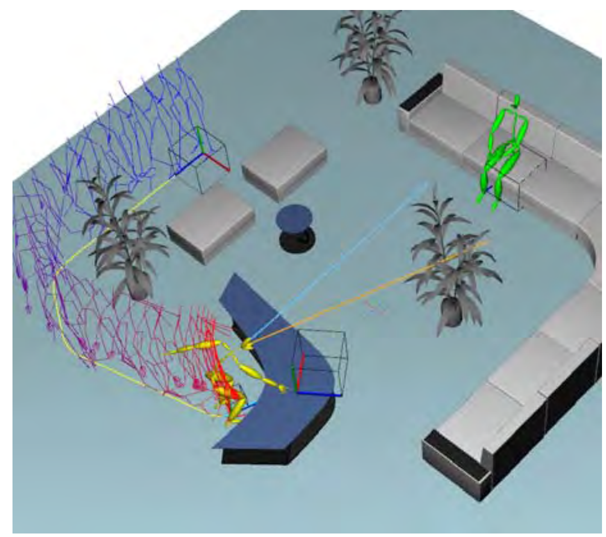
Planning Motions and Placements for Virtual Demonstrators
Yazhou Huang and Marcelo Kallmann
IEEE Transactions on Visualization and Computer Graphics (TVCG), 2015

Behavioral Model

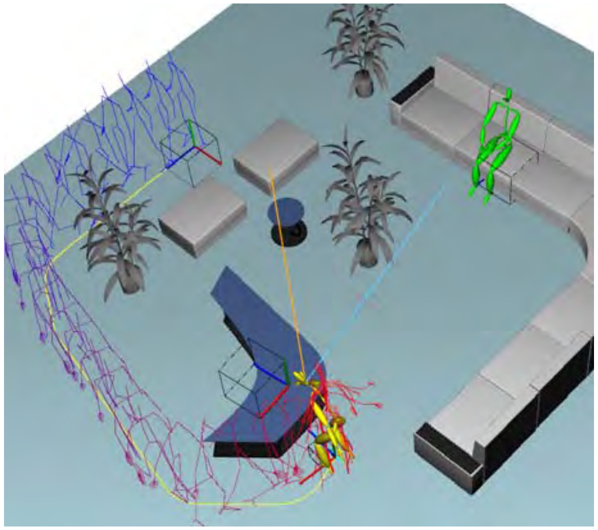
- Model derived from human subjects
  – 4 participants, actions to 6 objects, for 5 observers at different locations
    - Action: pointing and delivering info about the object



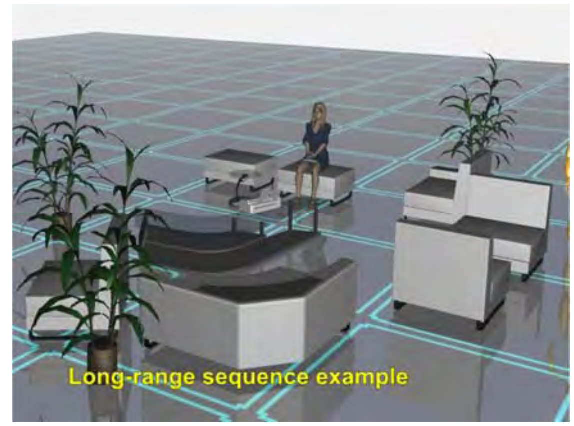
Placement Determination



Placement Determination



Additional Results



Planning Motions and Placements for Virtual Demonstrators
Yazhou Huang and Marcelo Kallmann
IEEE Transactions on Visualization and Computer Graphics (TVCG), 2015

- Acknowledgements
  - Grad students and Collaborators
    - Mentar Mahmudi, Yazhou Huang, Carlo Camporesi, Amaury Aubel
  - Funding Agencies
    - CITRIS Seed Funding (#12, #14)
    - National Science Foundation

    (IIS-0915665, BCS-0821766, CNS-0723281, CNS-1305196)

Additional Information

- Additional Material
  - SIGGRAPH course notes
  - Webpages of the authors:

    http://graphics.ucmerced.edu/

    http://www.cs.rutgers.edu/~mubbasir/
  - Recent book published by the authors:

Geometric and Discrete Path Planning for Interactive Virtual Worlds
Morgan & Claypool, 2016

Thank You !