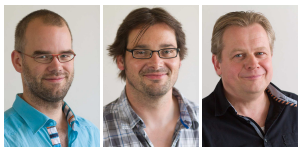
Universiteit Utrecht COMMIT/ COMMANDS NANYANG TECHNOLOGICAL UNIVERSITY

Collision Checking Using Occupancy Spaces



Sybren A. Stüvel
Arjan Egges
A. Frank van der Stappen

Virtual Human Technology Lab
Universiteit Utrecht
http://vhtlab.nl/

Nadia Magnenat-Thalmann
Daniel Thalmann

Institute for Media Innovation
Nanyang Technological University
http://imi.ntu.edu.sg/

Collision checking in crowds

Crowd simulation plays an ever increasing role in different fields, each having different requirements for their applications. Full collision detection on all characters is computationally expensive, hence for a real-time crowd an approximation of the character shape is needed.

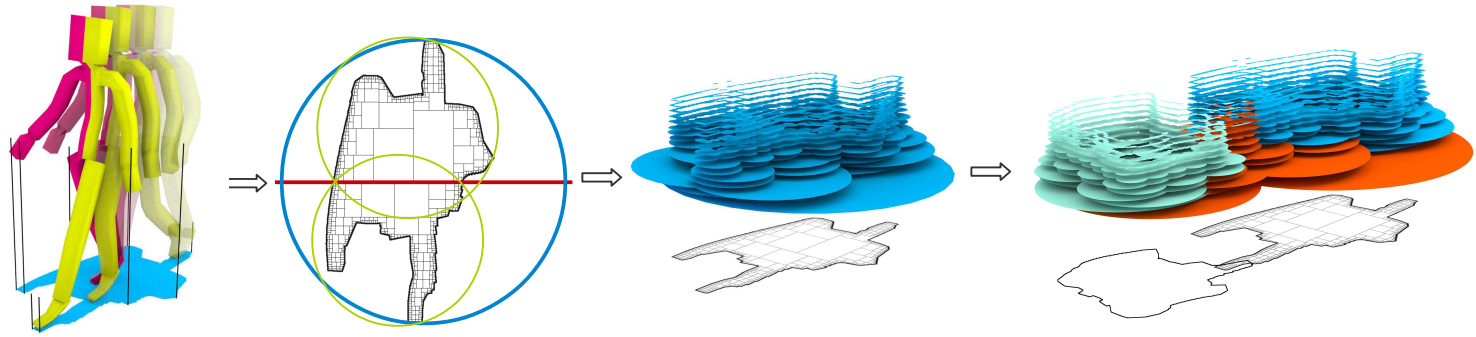
Many crowd simulation systems model a crowd agent as a cylinder. However, sometimes a more precise representation is needed.

Occupancy Space

We propose a novel method of modelling the space occupied by animated meshes, called the **occupancy space** (ospace).

The occupancy space OS(M, [t_1, t_2]) is defined as the ground projection of a character's animated mesh M over the time interval [t_1, t_2].

It thus forms a more precise approximation of the mesh shape than a bounding cylinder.



Motion segments
Project mesh onto ground plane

Bounding Circle Hierarchy
Representation of occupancy space allows for fast intersection tests

Intersecting occupancy spaces
Intersecting circles are shown in orange. Only a subset of those circles are visited during intersection tests.

Representation

To represent the ospace we propose a bounding circle hierarchy (BCH). For each ospace **OS(M, [t_1, t_2])**, the **smallest circle** enclosing the shape is stored. The algorithm then divides the shape into two halves, stores their two smallest enclosing circles, and recurses until the radius of the circle is smaller than a predefined threshold. The result is a binary tree where each node describes the centre and radius of the circle.

Using a threshold of 1 cm, an **intersection test between two BCHs takes around 17 µs on a current computer**.

The BCH was chosen in favour of other space partitioning techniques, such as a quadtree. Rotation is required to align an ospace with a character, and the circles provide us with a **rotation-symmetric representation**.

Scan for a digital copy
QR + NFC

Collision checks

```
intersect(BCH₁, BCH₂, T) → bool:
    if BCH₁ = ∅ or BCH₂ = ∅:
        return false

    if BCH₁.circle  ∩  T * BCH₂.circle = ∅:
        return false

    if BCH₁ has sublevels and BCH₂ has sublevels:
        return intersect(BCH₁.left, BCH₂.left, T) or
               intersect(BCH₁.left, BCH₂.right, T) or
               intersect(BCH₁.right, BCH₂.left, T) or
               intersect(BCH₁.right, BCH₂.right, T)
    if BCH₁ has sublevels:
        return intersect(BCH₁.left, BCH₂, T) or
               intersect(BCH₁.right, BCH₂, T)
    if BCH₁ has sublevels:
        return intersect(BCH₁, BCH₂.left, T) or
               intersect(BCH₁, BCH₂.right, T)

    Neither have another level, so this intersection
    determines the outcome.
    return true
```

Conclusion

We have implemented a crowd simulation using the above system, and observed that characters choose slower, smaller movements when available space becomes limited.

Note that **our method does not require *all* characters to use the same animation technique**. For example, one could use motion capture to drive a single character, and in real-time calculate and register the ospace that it occupies. A cylinder-based crowd system could also interact, since a cylinder is simply a single-circle BCH.

We are currently evaluating the chosen method of recursive intersection, and performing experiments to determine optimal settings for the representation and the algorithms that use it.