navi+

navigation mesh with collision shapes.

- 직관적인 논리로는 완벽
- 관건
  - 충돌 정보의 추출 (성능/효율)
    - TriMesh
    - 지형
  - 게임 플레이 데이터 화

개요

서버에서 공간 처리는 길찾기와 판정으로 이루어진다. 대부분의 게임들이 2D 형태의 그리드에서 판정을 진행한다. FPS처럼 충돌을 서버에서 처리하는 경우는 거의 없다.

Autodesk Navigation의 경우 네비매시 상에 여러 볼륨들을 설치할 수 있다. 이 정보를 이용하여 동적인 공간 속성의 변형을 처리할 수 있다. 또한, 우회하기를 제공하여 서로 겹치지 않도록 한다. 게임성이 많이 올라간다.

풀 3D 게임의 경우도 가끔 이상한 현상들이 보일 때가 있고 성벽을 부순다던가 하는 기능을 구현할 수 없는 경우도 있지만 서버에서 2D 형태로 거리 체크만 안정적으로 처리해도 현재까지는 문제가 없다.

여기서 조금 더 발전한 형태가 검은 사막에서 구현했다고 하는 복셀 기반 이동과 충돌 처리이다. 시선 차폐와 같은 인지를 할 수 있고 넓은 성벽을 부술 수 있으며 화살이 장애물에 부딪혀 떨어질 수 있다.

복셀 기반으로도 서버에서 처리하는 충돌 관련 기능이 많이 개선된다는 걸 알 수 있다. 하지만 검은 사막의 구현이 Greedy best-first 를 성능 향상을 위해 사용하면서 먼 길을 돌아가는 등의 문제가 있다. 또 복셀은 해상도를 올리게 되면 메모리 사용이 급격하게 올라간다. (물론 다양한 해결방법은 있다)

임의의 표면을 가장 적은 데이터로 잘 근사할 수 있는 방법은 다각형을 사용하는 것이고 렌더링에 사용되는 삼각형이 가장 다루기 쉽다. 또 어떤 모양은 박스나 구면 충분한 경우도 많다. 이와 같이 충돌 처리에 사용되는 기법을 데이터를 최적화하여 서버에서도 사용할 수 있도록 한다면 다음 단계의 상세한 서버 처리가 가능하게 된다.

복셀이냐 매시 기반이냐를 선택하기는 쉽지 않은 문제인데 가능하면 두 가지를 모두 구현해 보고 싶었다. 하지만, 언리얼 4 처럼 소스를 변경하거나 추가할 수 있는 경우가 아닌 유니티의 경우는 C#으로 중요하고 빠른 기능을 구현하여 맞춰야 하기 때문에 (클라 / 서버가 다른 알고리즘을 쓰면 수치 안정성이 떨어짐) 적합하지 않아 보였다. 다른 측면에서 복셀들을 잘 묶어서 정확하게 처리하려면 계산 기하학 (Computational Geometry) 공부와 연습이 되어 있어야 하는데 쉽지 않았다.

따라서, 현실적으로 접근하여 Unreal 4에서 사용하고 있고 Unity에서도 사용하는 것으로 보이는 Recast / Detour 기반으로 길찾기를 하고 충돌 정보의 간략화, 공간 분할과 처리에 집중하여 이동과 장애물 처리가 가능한 라이브러리 구현을 진행하려 한다.

목표

아래 기능을 가진 라이브러리를 구현한다.

- 충돌과 네비매시의 통합
- 동적인 충돌체
  - 네비매시의 통합
- 길찾기
  - 에이전트 크기 반영
  - 충돌 정보의 반영
  - 이동 형태의 반영
    - Walking
    - Flying
    - Jump
    - 복합 충돌체와 애님
- 충돌
  - 복셀 트리
    - 정적인 충돌 정보
  - AABB / OBB
  - Sphere
  - Cast
    - Ray
    - Sphere
    - Disk
- 공유
  - 정적인 충돌 정보
  - 네비 매시 정보
  - 동적인 정보만 분리
- 툴
  - Unity Exporter / Visualizer
  - UE4 Exporter / Visualizer

기반

- Recast / Detour
  - Height Value
  - way point 수준으로 사용
- Bullet
  - 넓은 지역의 공간 분할
  - 충돌 정보의 사용

진행

- 네비게이션
  - 네비 매시 정보 Export
  - 오프매시 링크 Export
- 충돌 공간
  - Export
  - Partition
- 통합
  - 길찾기 + 충돌
  - 지역 명령
  - 게임 연동 인터페이스
- 디버깅 / 모니터링 툴

네비게이션

UE4 내에 아이디어 구현. 이미 내장 되어 있으므로 서버처럼 뒤지면 된다. PhysX 와 Detour를 함께 사용하고 기존 코드를 분석한다.

Recast / Detour

- Detour 인터페이스
- Detour 알고리즘
- Recast 인터페이스
- Recast 알고리즘

생성

- 바닥 폴리곤들 추출
- 복셀로 변환
- 갈 수 있는 복셀들만 추출
- watershed로 복셀들을 영역으로 나눔
- 각 영역의 외곽선을 단순화
- 각 영역을 폴리곤으로 만듦
- 삼각형 연결 정보를 만듦

계산기하학에 대한 이해가 필요하지만 기본 아이디어는 이해할 수 있다. 복셀 변환된 부분을 정리해서 Voxel Tree를 만드는데 사용할 수 있다. 물론 contour 정보 등을 추가하고 효율적으로 만드려면 추가 노력이 필요하다.

충돌 정보

- 충돌 정보 Export
- 공간 분할
- 동적 에이전트 이동
- Collision Query
- 길찾기 연동
- 인스턴스간 정보 공유

Export

- StaticMesh
  - Complex
  - Simple
- Terrain Mesh or Terrain

간략화 알고리즘이 가장 중요하다. 비용을 대단히 낮추는 것이 필요하다.

- NaviMesh

지형의 사용 여부와 연관되어 있다.

이 부분의 핵심은 최대한 간략화된 정보를 사용하는 것이다. 쓸 수 있는 방법을 모두 동원해야 한다.

Partition

게임 플레이를 고려하여 빠른 쿼리가 가능해야 한다. BVH를 사용하고 성능을 측정하여 꽁수를 만든다. 예를 들어, 게임 플레이 객체들만 추출할 경우 별도의 BVH를 사용할 수 있다. 아마도 채널 같은 기능이 Bullet에도 있을 듯 하다. 가까운 친구들을 찾아 가는 방법도 있으면 좋다. (없으면 만들어야 한다)

이 부분이 사용성 측면에서 중요하다.

bullet은 Dbvh (Dynamic Bounding Volume Hierarchy. AABB 트리) 를 기본으로 한다. 서버는 에이전트를 빠르게 검색 가능한 공간이 필요한데 주로 Octree나 Quadtree가 적합하다.

순회나 에이전트 검색을 위해 메모리 효율적인 Octree를 충돌 공간 기반으로 생성한다. 네비매시 주변을 플레이 공간으로 볼 수 있는 게임도 있고 아닌 경우도 있으므로 파라미터로 처리한다.

Bullet을 완전히 둘러싸서 인터페이스를 만든다.

통합

- 에이전트 이동
  - 길찾기 + 충돌
  - Avoidance

- Area Command
    - 트래픽 전파 등
- 충돌 정보와 연결된 에이전트

자료

https://accu.org/index.php/journals/1838

- 네비게이션 매시 생성 알고리즘
- 좋은 논문. 좋은 연습문제들을 포함
- 미코모노넨 (Recast) 방법을 개선하는 게 좋을 듯 하다.

https://github.com/ncsoft/Unreal.js

- unreal javascript 연동 기능
- 스크립트 기능이 없는 언리얼에 매우 좋은 대안

https://github.com/drywolf/react-umg

- UI 개발 도구
- React 언어로 구성하면 UI 렌더링

https://github.com/charto/nbind

- nbind 라이브러리