Voxel+

voxel for pathfinding, collision, and tagging.

target games:

- RPG
- FPS

정리

이 방향으로 꽤 진행했다. 기본 엔진들에 통합하는 작업량이 많고 안정성이 떨어질 듯 하여 복셀 트리로 맵 정보를 충돌에 활용하고 네비매시로 바닥을 이동하는 오브젝트의 길찾기를 사용하려 한다.

빈 공간을 날아가는 경우 여기서 정리한 아이디어들을 활용할 수 있다.

목표

아래 기능을 가진 라이브러리를 구현한다.

- 충돌과 길찾기 통합
- 동적인 충돌체
- 길찾기
  - 에이전트 크기 반영
  - 충돌 정보의 반영
  - 이동 형태의 반영
    - Walking
    - Flying
    - Jump
    - 복합 충돌체와 애님
- 충돌
  - AABB / OBB
  - Sphere
  - Cast
    - Ray
    - Sphere
    - Disk
- 공유

    - 정적인 충돌 정보
    - 네비 매시 정보
    - 동적인 정보만 분리
- 툴
    - Unity Exporter / Visualizer
    - UE4 Exporter / Visualizer

알고리즘

- 1비트 배열로 전체 정적 voxel 트리 생성
    - 리프 노드들은 충돌 채널 정보 추가
    - 리프 노드들의 contour 평면 정보 추가
- Cell
    - 복셀 트리와 같은 구조
    - k 레벨에서 분할
    - 그래프 구조로 연결
    - 종류
        - walkable
        - flying
- 충돌
    - voxel tree와 동일 범위 octree 사용
    - 레벨을 voxel tree보다 작게 함
    - 충돌 알고리즘 기반
        - bullet 등의 코드 재사용 또는 포팅

배경 아이디어

게임 월드를 보면 매시 형태의 충돌 처리가 많다. 매시는 폴리곤 단위 충돌 처리가 많아 서버에서 수용하기 적합하지 않다. 따라서, 복셀로 대략적인 충돌 처리를 하는 것이 적합하다. 동적인 오브젝트는 기존과 같이 캡슐, 실린더, 구, 캐스팅 (Ray, Disk)을 사용한다. 복셀은 AABB에 해당한다.

네비매시를 사용하는 방법은 walkable 엔티티에만 적용되는 공간 정보이기 때문에 일반화되기 어렵다. Cell로 walkable 복셀들을 묶고 그래프로 연결하면 네비매시와 동일한 기능을 제공 가능하다. 날아서 이동 가능하는 엔티티를 위해 별도의 flying 셀들을 만들 수 있고 크기가 다른 엔티티들에 대해 별도의 walkable cell들을 만들 수 있다는 있다는 점도 장점이다.

스트리밍

대규모 레벨에 대해 stream in / stream out 이 가능하도록 구성한다.

복셀 자료 구조

k (<n) 레벨의 복셀 트리는 비트로 하고 마지막 리프 레벨 데이터에 대한 leaf block index를 갖는다. 0으로 채워지지 않은 공간은 리프를 필요로 하지 않으며 1인 경우만 상세한 복셀 정보가 필요하다.

Leaf

- solid or empty (1bit)
- contour (15bits)

Node:

- filled or empty (1bit)

Holder : Node

- block index (16 bits)
- offset (16 bits)

Holder 가 갖는 Leaf의 개수와 공간 내 위치는 미리 파마미터로 결정한다. 이 파라미터들에 의해 Leaf 노드를 얻을 수 있다.

셀 자료 구조

복셀 부분은 동일하다. 충돌 채널과 속성만 추가한다. attribute block에 대한 인덱스로 한다.

세부 구현

충돌 복셀 트리

목표

UE4를 대상으로 진행한다. 충돌과 매시 정보를 활용한다. FNavGenerator를 참조하여 복셀을 추출한다. contour 정보를 포함한다.

생성된 정보를 UE4 내에서 렌더링한다. 색상과 투명도를 조절할 수 있도록 한다.

분석 (FNavGenerator)

Recast의 네비 생성 언리얼 내 구현을 살핀다. 이동 처리를 통합하는 게 쉽지 않을 듯 하다. Navigation과 복셀 트리를 함께 쓰는 방식을 고려한다.

Unity와 Unreal 4 모두 네비매시 기반으로 길을 찾고 충돌을 처리하면서 이동하는 방식을 사용한다. 충돌은 물리 엔진을 사용한다. 서버는 매시를 포함하는 물리 정보를 처리하기에 부담이 되기 때문에 이를 정적인 복셀 정보와 동적인 간략한 모양들로 처리하는 방식이 적절해 보인다.

논문 - 복셀 기반 실내 길찾기

석사 학위 논문이다. 기본적이긴 하나 프로토타잎을 진행하기 위한 기반으로 괜찮아 보인다. 언리얼 4의 맵에서 복셀 정보를 추출하고 Cinder 기반으로 렌더링하고 테스트 해본다. 여러 가지 파라미터를 측정해서 방향을 확정한다.

알고리즘

Dijkstra, A*, breath first, depth first 를 리뷰한다.

Bushfire는 시작 점부터 enumeration level을 증가시키면서 이웃들을 추가한다. 이웃들은 expansion을 통해 확장되므로 많은 노드들이 포함된다. 아이디어 수준으로 추가된 것으로 보인다.

Greedy best-first search는 목표 지점까지의 거리만 기록한다. 이 값에 의존해서 목적지에 가까운 노드들만 추가한다. 하지만 가장 짧은 길을 보장하지 않는다.

A*는 bushfire와 greedy best first를 섞어서 쓴다. 휴리스틱 함수 $f(n) = g(n) + h(n)$ 으로 시작 점까지의 거리인 $g(n)$ 과 끝점 까지 거리인 $h(n)$을 함께 사용한다.

A*는 장점에도 불구하고 메모리 사용량이 늘어나고 반복해서 찾아야 하는 문제가 있다. 이를 개선하기 위해 계층적인 구조를 필요로 한다.

복셀 대신 Voxel Tree로 구성하려는 계획이다. 여러 가지 상세한 구현 수준에 따라 품질은 달라진다. 화이팅.

Distance Transformation

바로 읽고 이해하기는 어렵다. n 차원의 이미지에서 거리를 미리 계산하는 방식으로 가장 가까운 길을 찾는다. 나중에 한번 더 보도록 한다.

Configuration Space

로봇 공학에서 온 용어. 액터의 자유도 (관절 등에 기인한)에 따른 상태 공간. 이 공간에서 충돌 없는 길찾기가 가능하면 길을 찾을 수 있다. 수학적인 변환이 필요한데 차원이 높아지므로 복잡도도 같이 올라간다.

그래프 생성

필라 기반. 네비매시 생성 알고리즘 중 하나. Xiong et al., 2015년 논문. 미코노넨 구현의 이론 기반.

거리 필드 기반. Cell과 포털 생성해서 그래프로 만듦.

HPA* (Hierarchical Path-finding A*) --> 2D만?

Methodology

- Zlantanova et al., 2014
- Cell decomposition. Haumont et al., 2003
- Graph generation. Botea et al., 2004

Semantic labels to the empty space.

Cylinder. Mode of locomotion : driving, walking, flying.

Cells assigned to semantic classes : floor, stairs, obstacls.

Voxelized model --> Dilation --> Semantic Labeling --> Cell generation --> Graph Generation --> Path Finding

Dilation : 팽창.

Input Parameters

- Actor diameter (width and length)
- Actor height
- Mode of locomotion
- Vertical footspan
    - 상하로 움직일 수 있는 범위 (다리 올리기)
    - 점프도 이걸로 수용 가능?

Dilation

동적으로 변화하거나 크기가 다른 여러 액터를 지원하지 못 한다. 크기를 고려해서 찾을 방법은 없는가?

Semantic Labelling

- Extraction of horizontal surfaces
- Segmentation of horizontal surfaces
    - 제일 낮은 곳에서 물 채우기로 세그먼트 분리
    - 같이 채워지면 같은 세그먼트. 아니면 분리
    - Vertical footspan을 반영하여 추가
- Slope estimation
    - 이웃 간에 평면을 만들고 기울기 측정
- Upwards propagation of classes

Cell Generation

Distance transformation --> Watershed transformation --> Cell compression --> Cell merging

거리 변환 또는 거리 필드를 보고 다시 읽는다.

가장 가까운 Geometry까지의 거리를 사용하는 변환. --> 셀 생성에 사용.

좀 더 잘 이해되지만 아직 완전하지 않다.

Graph Generation

VoxelizedModel - Cell - Portal - Node / Edge - Graph

Path Finding

셀도 박스 형태라고 보고 위치를 찾아서 셀 그래프에서 가장 짧은 경로를 먼저 찾은 다음 셀 내에서 경로를 찾는다.

구현

ParaView와 Python을 사용.

ParaView는 VTK 기반. VTK의 내부 구현은 확장이 어렵다. Visualization에 맞춰짐.

연습

- 2D 그리드 맵 navigation
- 2D 그리드 맵 충돌 처리
- 3D 복셀 맵 생성과 길찾기
  - Naive
- 3D 복셀 맵 생성과 계층화
  - Cell?

Distance Field

https://imagej.net/Distance_Transform_Watershed

- Distance Transform watershed

공간은 이미지의 특성을 갖는다. 또는 이미지가 공간 특성을 갖는다. 둘은 상호 연결되어 있다. 거리 필드도 공유된 느 특성 중의 하나다. 공간이나 이미지의 특성을 수학으로 파악한다.

논문 1. Distance Fields for Rapid Collision Detection in Physically Based Modelling

https://pdfs.semanticscholar.org/ec41/48aed023dfe1ba6f42a198613800fe29ae37.pdf

연속적인 충돌 문맥에 거리 필드를 사용한다.

Distance Field Computation

표면 S위에 정의된 스칼라 함수이다. R^3 공간 전체의 점들에서 정의되고 S 까지 최소 거리가 함수 값이 된다.

$D : R^3 \rightarrow R$

$D(p) = min_{q \in S}\{|p - q|\}, \forall p \in R^3$

내부와 외부를 구분하기 위해, 부호를 도입한다. 내적을 사용하여 정의한다.

$sgn(p) = -1 if < p - q|n >< 0$

$sgn(p) = 1 if < p - q|n >\geq 0$

대략의 알고리즘은 다음과 같다.

- 매시의 각 삼각형에 대해 프리즘을 면 법선 방향으로 꼭지점을 늘려서 만든다.
- 프리짐을 둘러싸는 축에 정렬된 정수 박스를 결정한다.
- 정수 박스의 각 그리드 점들에 대해 삼각형까지 거리를 계산한다.
- 특정한 점과 삼각형까지의 거리를 보로노이 영역을 찾아서 빠르게 계산할 수 있다.

잘 모르는 점은 그리드 점 (grid points), Voronoi regions 이고 실제 구현을 할 만큼 이해하지는 못 했다.

그래도 기본 아이디어는 따라간 것 같다. 특히, 거리 필드가 뭔지 알 게 되었고 매시의 삼각형에 대해 박스를 만든 후 이 값을 대략 빠르게 계산할 수 있도록 했다는데까지는 이해했다.

이후 전개는 설명은 아니므로 생략한다.

논문 2. 알고리즘 비교

- 이미지의 경계를 찾아야 거리를 계산할 수 있다.
    - Immediate Interior (II) / Immedidate Exterior (IE)

오브젝트에 해당하는 점일 경우 일정 범위 내에 있는 점들 중 배경인 점이 있으면 경계에 해당하고 배경에 해당하는 점일 경우 일정 범위 내에 있는 점들 중 오브젝트 점이 있으면 경계에 해당한다.

느슨하지만 대부분의 경우 이렇다고 가정할 수 있다. 증명 가능한가?

단순한 거리 변환 알고리즘

경계에 0을 할당. 다른 점들은 경계점들과 거리 중 최소값을 선택.

단순하지만 계산량이 매우 많다. 아이디어가 그렇다는 뜻.

개선 버전

Danielsson's distance transform.

- 경계에 거리 0을 할당, 나머지는 무한대를 할당
- 변환되는 이미지 I' 에서 여러 번 처리 (지역적인 범위에서)
- 현재 거리를 고려중인 이웃들의 거리와 이웃과 자기 간의 거리를 비교해서 더 작으면 교체

무한대에서 점점 여러 번 이웃의 거리를 고려하여 줄여 나가는 방식.

예제 알고리즘들은 y를 늘려가면서 한번 줄이고 y를 감소시키면서 한번 더 조정한다.

이해한 바의 정리

경계를 찾는 부분은 아미지 인식에 사용할 수 있다. 배경도 경계까지의 거리를 갖고 있다면 충돌과 길찾기에 사용할 수 있다. 거리 값이 크면 안심하고 가도 되고 작으면 충돌이 가까워진다는 뜻이다.

동류를 엮는 방법으로 사용할 수 있다. 이미지의 0/1 값이 아닌 속성의 0/1 값으로 하면 같은 종류로 묶을 수 있다. 0/1 값을 어떤 다른 수치에서 0/1이 되도록 하면 좀 더 많은 속성 값 범위에서 분류할 수 있다.

아이디어는 단순하지만 응용은 다양하다.

WaterShed

유역. 물이 흐르는 영역. Cell로 나눌 때 distance field와 함께 사용하는 방법이다. 관련 아이디어를 찾고 추가로 이해한다.

https://pdfs.semanticscholar.org/766f/b0fe558952090c65c21151677706532e2a12.pdf

이미지 분할 계열이다.

morphological : 형태학의. morphology가 형태소 분석.

특정 구조에 극소값에서 물을 채우기 시작하고 극대값에서 멈춘다. 극대가 경계가 된다.

뭔가 쓰다만 논문일세. 대략의 과정만 보여준다.

http://disp.ee.ntu.edu.tw/meeting/%E6%98%B1%E7%BF%94/Segmentation%20tutorial.pdf

튜토리얼 문서로 4절에 워러쉐드 내용이 있다.

묙표가 이미지에 있는 watershed 라인을 찾는 것이라 나온다. 아마존 같은 큰 강은 지도나 위성 사진에 라인으로 나온다. 이런 라인은 주변의 높이에 따라 형성된다.

물을 채우면서 나누는 선 (Watershed)를 찾는다. 알고리즘이 상세하게 구현이 가능한 수준으로 설명되어 있다.

너무 잘게 나눠지는 문제가 있어 마커를 사용해서 부드럽게 만든다. 길찾기나 충돌 처리용은 좀 나눠져도 크게 문제 없을 듯 하다.

Sparse Voxel Octree

렌더링 관련 논문이다. Contour 정보 처리 방법과 자료 구조를 활용 가능할 지 확인하기 위해 본다.

http://www.nvidia.com/docs/IO/88972/nvr-2010-001.pdf

Surface에 대해서만 voxel 데이터를 갖는다. 이를 통해 메모리 사용량을 줄일 수 있다.

복셀 자료 구조

Blocks

- array of child descriptors
- info section
- contour data
- attachments
    - shading attributes

차일드 위치를 나타내는 16 비트 (far 비트를 포함). near / far 를 사용하여 메모리 블럭을 사용한다. 블럭은 페이지 단위로 나뉘어 있다.

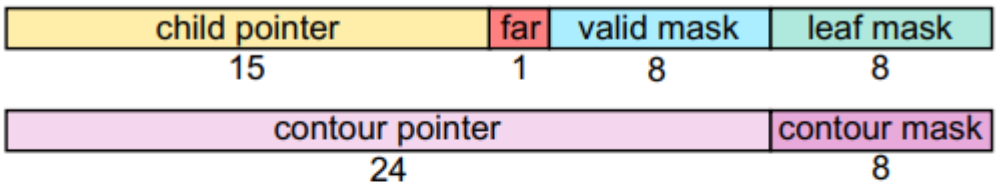

Figure 2: *64-bit child descriptor stored for each non-leaf voxel.*

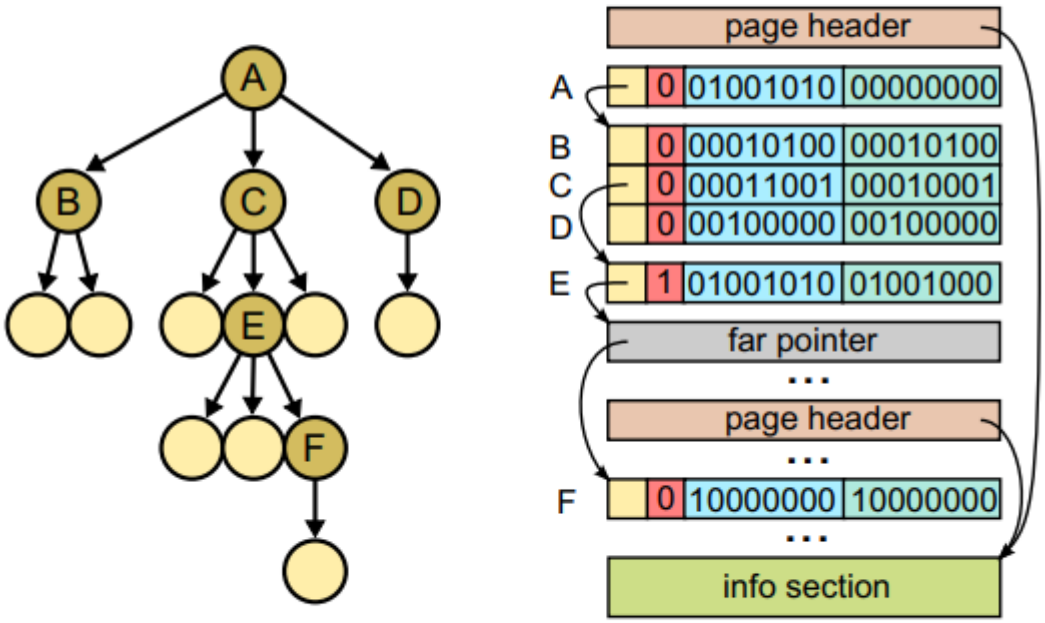

Figure 3: *Layout of the child descriptor array. Left: Example voxel hierarchy. Right: Child descriptor array containing one descriptor for each non-leaf voxel in the example hierarchy.*

컨투어 정보는 두께, 위치, 노멀 값으로 이루어진다.

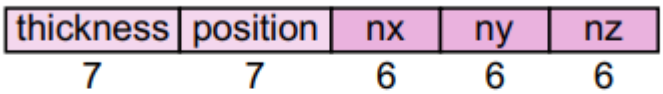

Figure 4: *Bitfields in a contour value. Components of the normal vector as well as spatial position are encoded as signed integers. Thickness is encoded as an unsigned integer.*

렌더링에 필요한 정도의 자세한 정보는 필요 없으므로 position과 nx/ny/nz 정도면 충분하다.

위치는 27개 정도로 복셀을 나누고 노멀은 16단계의 회전 값으로 하여 1바이트 내에 포함한다. 위치 5비트, 노멀 4/4/4 (12) 비트 정도면 충분할 것으로 보인다. 리프 노드만 이 값이 필요하다. 완전히 채워진 플래그를 1비트 사용한다.

자료

https://www.youtube.com/watch?v=UokjxSLTcd0

- 비슷한 아이디어 계열

http://www.critterai.org/projects/nmgen_study/heightfields.html

- 공부해서 정리한 내용
- 여기도 참고할만. 코드도 있으니.