

A close-up, top-down view of a large pile of colorful plastic balls. The balls are in four primary colors: red, yellow, green, and blue. They are densely packed together, filling the entire frame. The lighting is bright, creating highlights and shadows on the smooth, glossy surfaces of the balls. The text "Ball Pit" is centered over the image in a white, sans-serif font.

Ball Pit

프로젝트 목표

- Ball Pit을 구현
- Ball 하나를 Particle 하나로 취급
- 아주 간단한 물리 엔진으로 충돌한 Ball간 상호작용
- 제한된 공간에서 수만 단위의 Ball을 시뮬레이션
- 각 Ball에는 기본적인 조명과 텍스처 적용

최대의 어려움 = Ball 간의 교차연산

Naïve한 방법 :

- 10000개의 Ball(Particle)을 시뮬레이션 하기 위한 사이클 당 10000c2 번의 교차연산 필요.
- $10,000 * 9,999 / 2 = 49,995,000$ 번.
- 초당 60 사이클을 목표로 한다면 1초에 30억번의 교차연산을 해야함.
- 계산 복잡도 $O(n^2)$ 에 해당.

공간자료구조

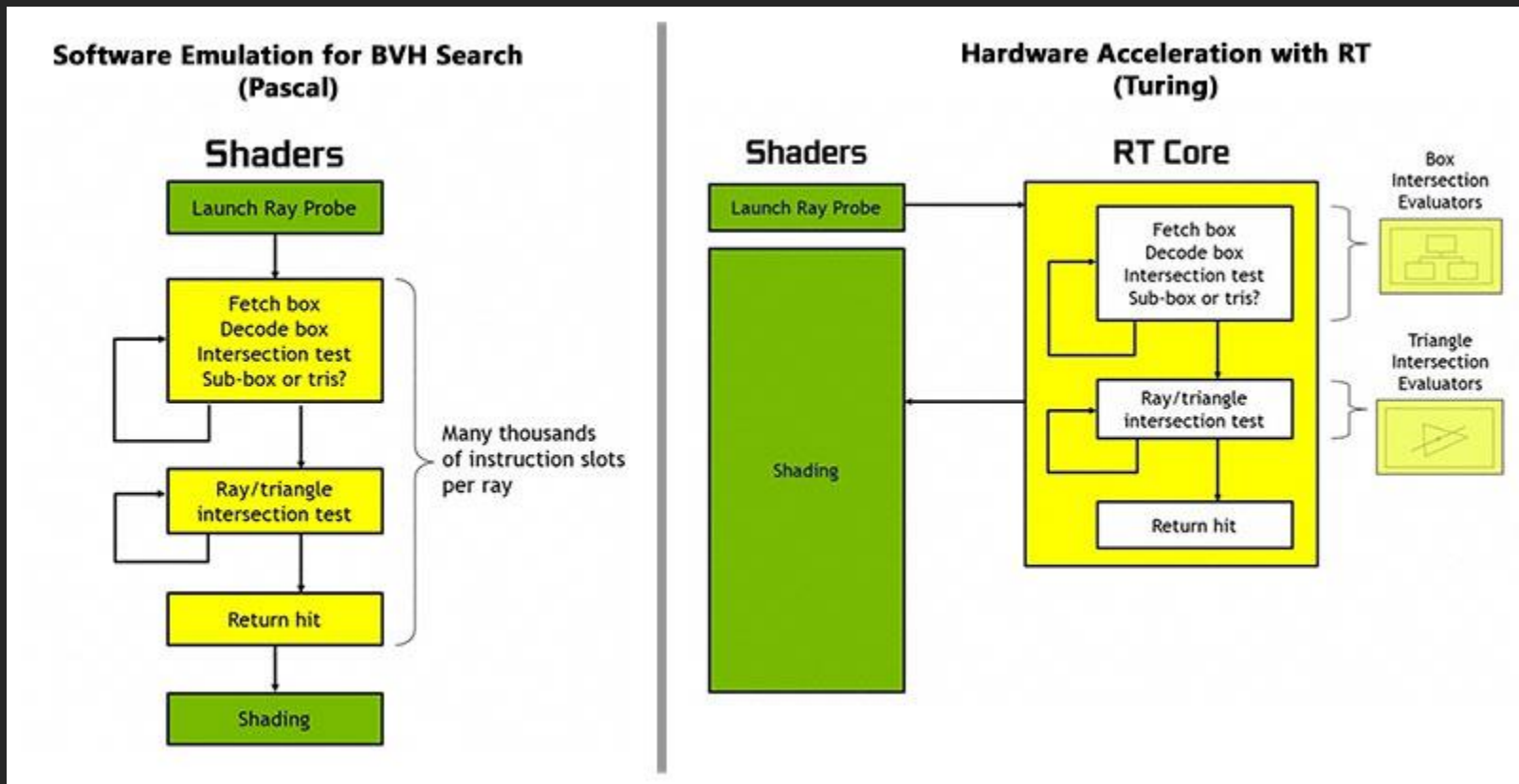
- 공간자료구조를 활용하면..
- $O(n^2)$ 문제를 $O(n \log n)$ 문제로 만들 수 있음.
- 교차연산을 사이클 당 백만 번 이하로 낮출 수 있음.
- 공간자료구조 : Octree, BVH, BSP등이 있음.
- 단 사이클 당 자료구조 구축 혹은 정렬에 필요한 약간의 추가 오버헤드

공간자료구조의 활용

공간자료구조는 현대 그래픽스에서 굉장히 중요한 위치

- 렌더링 최적화
- 리얼타임 파티클 시뮬레이션 가속
- 리얼타임 레이트레이싱 가속

대표적인 사례 - NVIDIA RTX



- RT Core는 실시간 레이트레이싱을 가속하기 위해 공간자료구조 순회와 교차연산만을 수행하도록 설계된 하드웨어임.

프로젝트 진행계획

- Library 없이 GPU에서 작동하는 공간자료구조 구현 시도. (OpenGL 사용)
- 실패하면 Library를 도입 후 재시도. (Nvidia PhysX등)
- 이도 저도 안되면 CPU에서 작동하는 코드로 구현.

```
GenerateSpatialTree() COMPLETE!  
Total : 40194
```

```
0.010677
```

```
GenerateCollisionTable() COMPLETE!  
Total : 51591  
Total : 51591
```

```
0.008337
```

(본인이 옛날에 만든 CPU코드 결과)

만약 제대로 GPU성능을 활용하는 공간자료구조를 구현한다면 매우 빠르게 동작할 것임.
CPU 코어 하나만을 이용하는 코드도 만육천개 파티클의 공간자료구조 및 충돌 테이블을 완성하는데 0.02초 미만이 걸렸음.

- 공간자료구조 구현에 성공하면 이를 기반으로 간단한 물리엔진 구현.
- Ball 간 충돌 시 서로 밀어내는 효과 등.
- 조명 및 텍스처를 적용하여 마무리.

끝