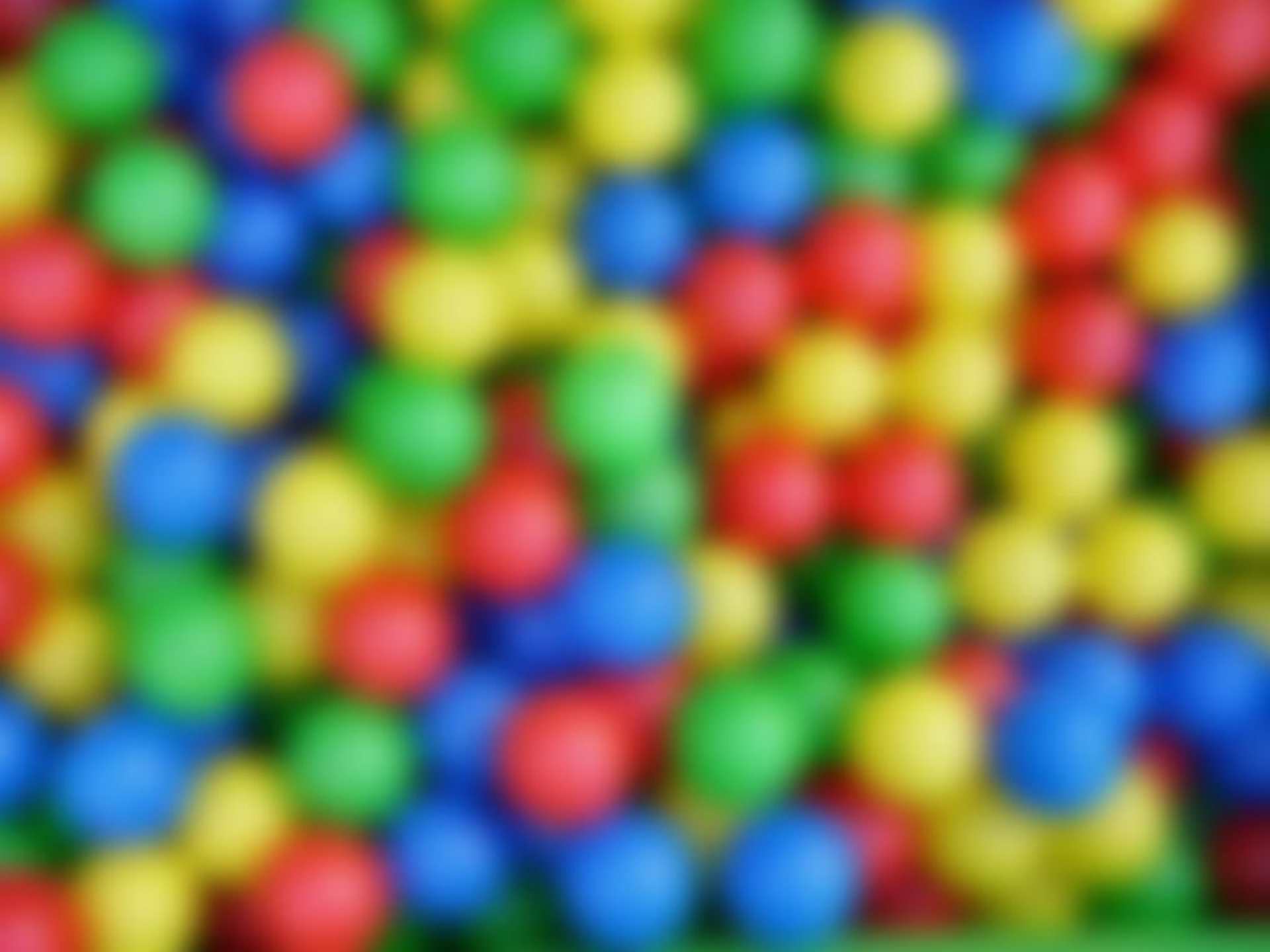



Ball Pit!

프로젝트 개요

- 프로젝트 주제 : Ball Pit!
- 프로젝트 내용 : Ball Pit! 구현
- 구현 플랫폼 : C++, OpenGL
- 사용 라이브러리 : Vector Math Library
  외 전부 본인 코드

큰 흐름

```
Main(){
    initialize();
    while(isExit){
        peekMessage();
        update();
        render();
    }
    terminate();
    return 0;
}
```

물리엔진

- 각 Particle들의 위치 및 속도 업데이트는
- 이전 상태 즉 한 사이클(프레임)전의 상황만 고려함

- 이 방식은 모든 Particle들의 업데이트를 독립적으로 할 수 있게 함으로 병렬화에 아주 유리하고 Computer Graphics에서 주로 사용됨.

- Particle Based Physics는 FEM(유한요소법)등의 방법에 비해 정확도가 떨어지는 것이 단점이나 Particle의 수와 갱신 빈도를 높일 수록 정확도를 올릴 수 있음

물리엔진

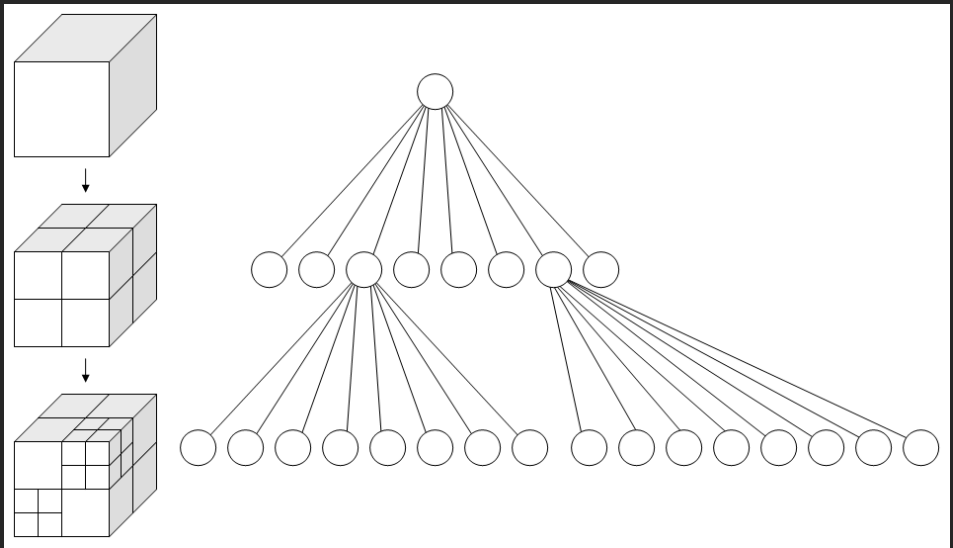
- 예상 외로 프로젝트 진행에서 가장 어려운 부분이었음.
- 굉장히 다양한 솔루션이 있음.
- 본인은 그 중 가장 간단한 것을 선택함.

- Particle(Ball) 간의 상호작용만 다룸.
- Spring, Damping, Shear Force
- 질량, 회전(각속도),공기저항 등은 고려하지 않음.

- 컴퓨터 그래픽스의 격언에 따라
  실제 물리현상을 그대로 묘사하기보다는
  보기에 그럴듯하도록 하는 것을 우선으로 구현.

공간자료구조 – 초기의 접근

- 프로젝트 구상단계에서 Particle Simulation을 가속하기 위한 자료구조로 Octree를 생각 했었음.
- Octree는 흔히 배우는 Binary Search Tree를 3차원 확장한 것임.



- Octree를 사용하면 3차원 공간에서 각 요소의 탐색을 아주 빠르게 할 수 있음.
- 내용이 조금이라도 바뀌면 새로 구성해야 하므로 사이클 당 한 번의 자료구조 생성 오버헤드가 존재.

더 빠른 방법 - VOXEL

- 메모리가 충분하다면 VOXEL을 사용할 수 있음.
- 미리 제한된 공간을 적절한 수의 격자로 쪼개 놓고 격자 수만큼의 Bin을 준비(희소 VOXEL이라는 방법도 있으나 어려운 주제로 본인은 시도하지 않음)
- 각 Particle들은 자신의 위치에 해당하는 Bin에 들어감
- 탐색할 때도 원하는 위치의 VOXEL만 확인하면 됨.

- 생성 및 탐색 계산 복잡도 O(n)!

- 본인의 프로그램은 2^18개의 VOXEL을 생성했으나
- 2^18 * 8(각 bin의 크기) * 2(Short Type Index의 크기) = 4MB만을 사용
- 여기서 2^10배 정도는 가볍게 늘릴 수 있음.

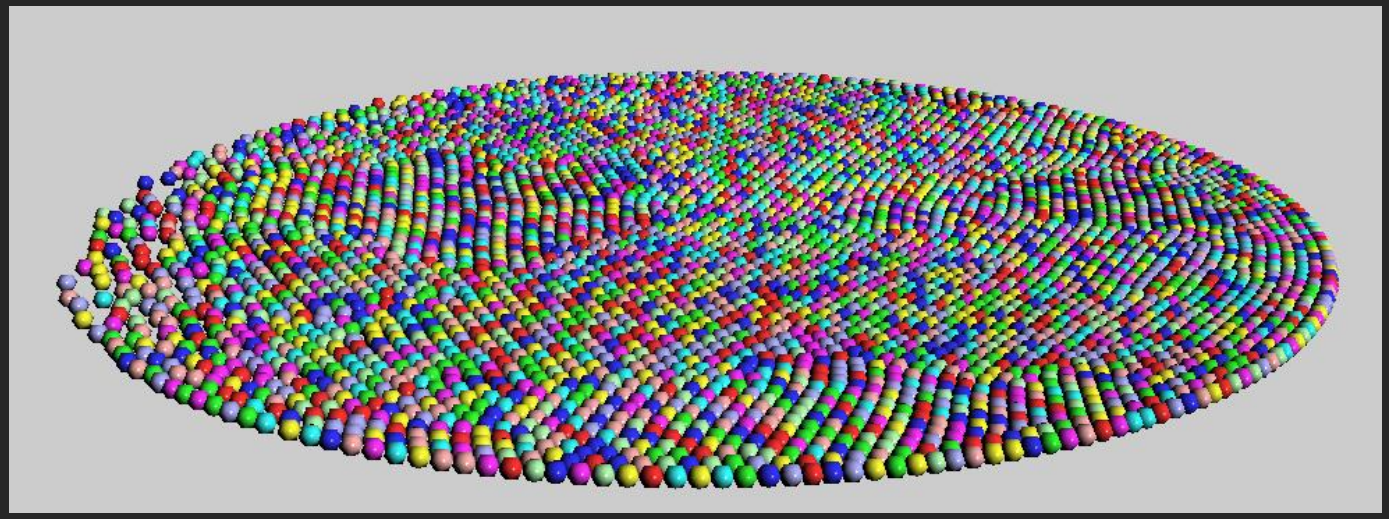
- 메모리가 충분한 현대에 자주 쓰이는 방식의 해법.

그래픽스

- GLSL을 사용한 간단한 단계의 Shader 프로그래밍.
- OpenGL의 편의 함수를 쓰는 것보다 성능 상 이점.

- Shader를 잘 응용하면 다양한 효과 구현 가능
  - 추후 과제

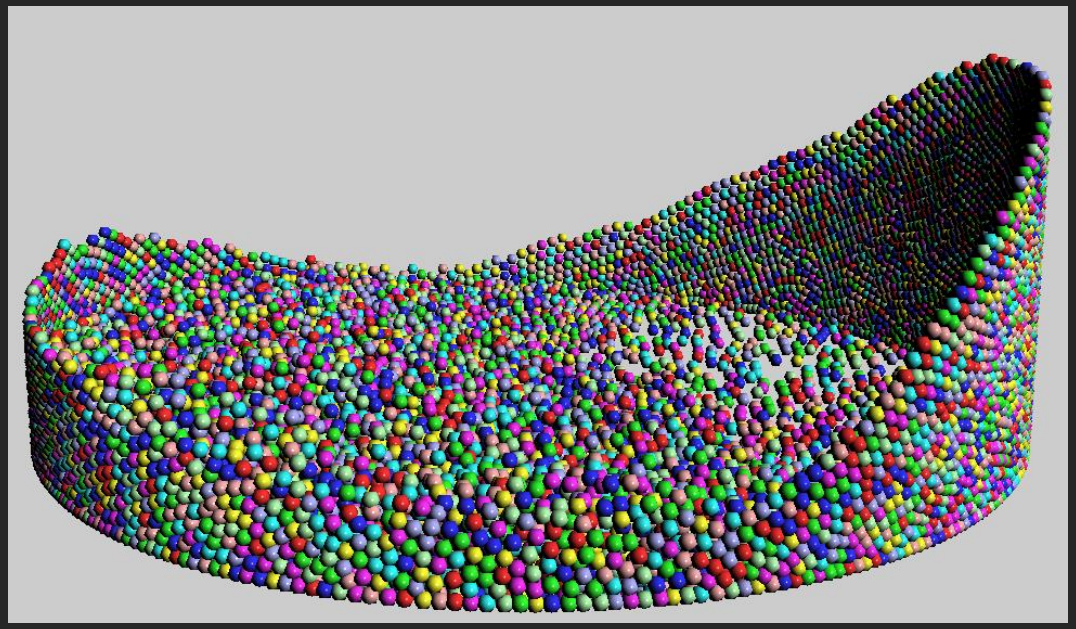
해결된 문제#1 – 중력



- 본인 프로젝트의 물리엔진은 강체 기반이 아님.
- 즉 각 Ball들이 근접하면 서로 밀어내기만 할 뿐, 교차가 가능함.
- 서로 밀어내는 힘보다 중력이 강하면 이런 현상이 생김.

- 주변에 충돌하는 Ball들의 수에 따라 일정량의 부력을 적용하여 중력과 반대 방향의 힘을 갖도록 하여 해결.

해결된 문제#2 - Ball Wall



- Ball들이 구체임에도 불구하고 서로 쌓여서 벽을 이루고 있음. - 현실에서는 벌어지기 힘든 일
- 온갖 변수가 있는 현실과는 다르게 본인의 물리엔진은 매우 간단한 법칙만으로 이루어져 있음. 즉 진동이나 기류 등의 요인이 없으므로 거의 완전한 구형임에도 서로 쌓일 수 있음.

- 가상의 진동을 넣어서 해결
- 매 사이클마다 난수를 생성하여 임의 방향으로 미세한 중력 적용
- Ball Wall이 생겨도 금방 무너져 내림.

남은 문제들 (추후 도전과제)

- 코드의 복잡함, 구조적이지 않은 코드
  - 시간이 모자랐음
  - 본인의 내공이 부족
  - 제출한 코드는 최대한 정리하였고 간단히 주석을 달았음.

- 물리연산의 Fidelity(정확도)가 초당 Frame(Cycle) 수에 의존
  - 초당 Frame 수가 낮아지면 Simulation의 정확도가 급격히 떨어짐.
  - Update함수와 Render함수를 분리해야 할 것.

- 현실세계 물리 법칙과 거리가 있음.
  - 실제 중력가속도는 9.8m/s이나 본인 프로그램은 0.01m/s에서 0.1m/s 사이.
  - 단순화한 물리엔진 때문으로 상기한 문제 외에 다른 종류의 Particle Simulation을 하기에 범용성이 부족

- 계산자원 활용 부족
  - GPU는 커녕 CPU도 단일코어만 사용하는 프로그램
  - 시간과 내공의 부족

끝