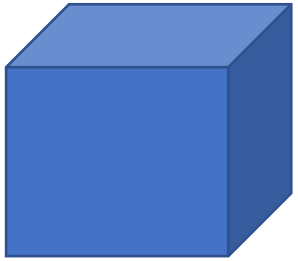
네트워크 스트리밍을 위한 복셀 데이터 압축

유영천

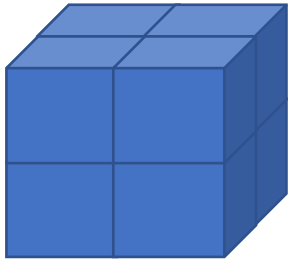
https://megayuchi.com

복셀 자료구조

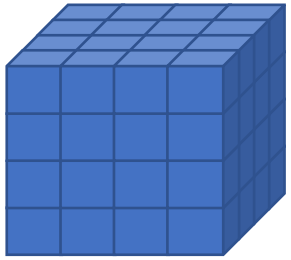
- 오브젝트 한개당 1x1x1 – 8x8x8까지 가변 정밀도
- 오브젝트 한개의 크기는 4m x 4m x 4m
- 기하구조 데이터와 텍스쳐 인덱스 데이터는 분리함.
  - (Vertex정보에 텍스쳐 인덱스 데이터 포함하지 않음)
- 기하구조는 복셀당 1 Bit, 최대 8 x 8 x8 1 bits = 512 bits = 64 bytes
- 컬러값(텍스처 인덱스)은 8 Bits, 최대 8 x8 x8 x 1 byte = 512 bytes



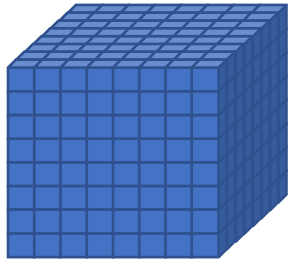
1x1x1 -> 1 bit -> 1 Byte



2x2x2 -> 8 bits -> 1 Bytes



4x4x4 -> 64 bits -> 8 Bytes



8x8x8 -> 512 bits -> 64 Bytes

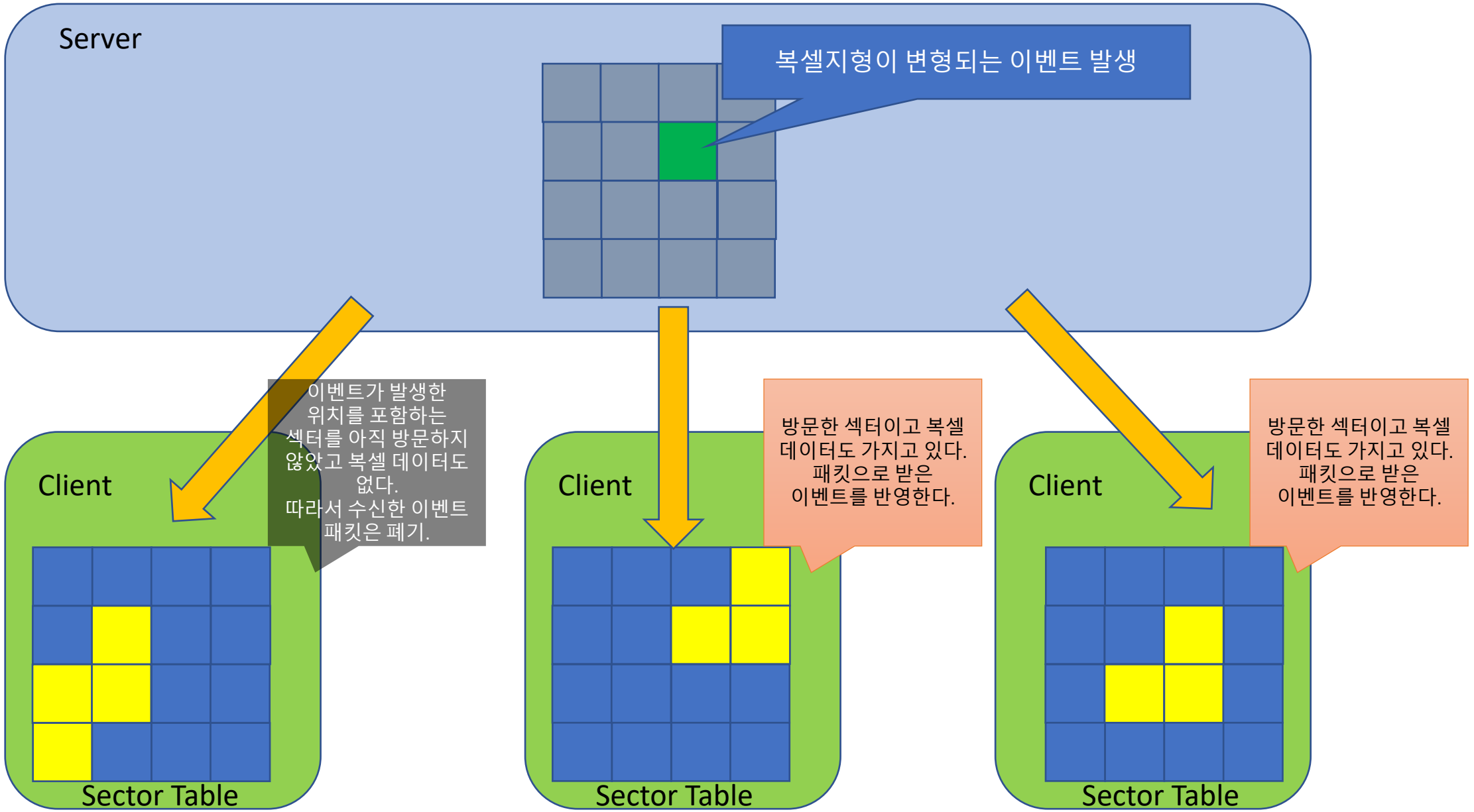
네트워크 전송 시나리오

- 로그인 후 맵에 처음 입장
  - 클라이언트는 복셀 데이터를 가지고 있지 않다.
  - 보고 있지 않을 때 어떤 변화가 생겼을지 모르므로 가지고 있을 수 없음.
  - 따라서 대량의 복셀 데이터를 수신해야함.
- 편집 이벤트
  - 복셀 오브젝트 추가 – 최소한의 파라미터와 이벤트만 전송
  - 복셀 오브젝트 제거 – 최소한의 파라미터와 이벤트만 전송

복셀 데이터 전송 규칙


Time
Server
Player #1
Sector Table
Player #2
Sector Table
Player #3
Sector Table
Server
Player #1
Sector Table
새로운 섹터로
이동했으므로 진입한
섹터의 복셀
데이터를 전송
Player #2
Sector Table
방문했던 섹터로
이동했으므로
복셀데이터를
전송하지 않음
Player #3
Sector Table
새로운 섹터로
이동했으므로 진입한
섹터의 복셀
데이터를 전송

복셀 데이터가 변형되는 이벤트가 발생했을 때 패킷 처리



섹터간 이동시 패킷량 변화

- 리소스 모니터로 확인

복셀 데이터 압축

- 온라인 모드에서 클라이언트는 복셀 데이터를 가지고 있지 않다.
- 복셀 데이터를 미리 가지고 있을 수도 없다.
- 따라서 맵에 최초 입장시 대량의 패킷을 송수신하게 된다. (9섹터 분량의 복셀 데이터를 수신해야한다)
- 서버 -> 클라이언트 패킷 버퍼가 모자라는 상황이 실제로 발생
- 기하구조 압축 필요
- 컬러 테이블 압축 필요
- Zlib등 범용 압축기법은 압축률이 너무 낮고 너무 느리다.

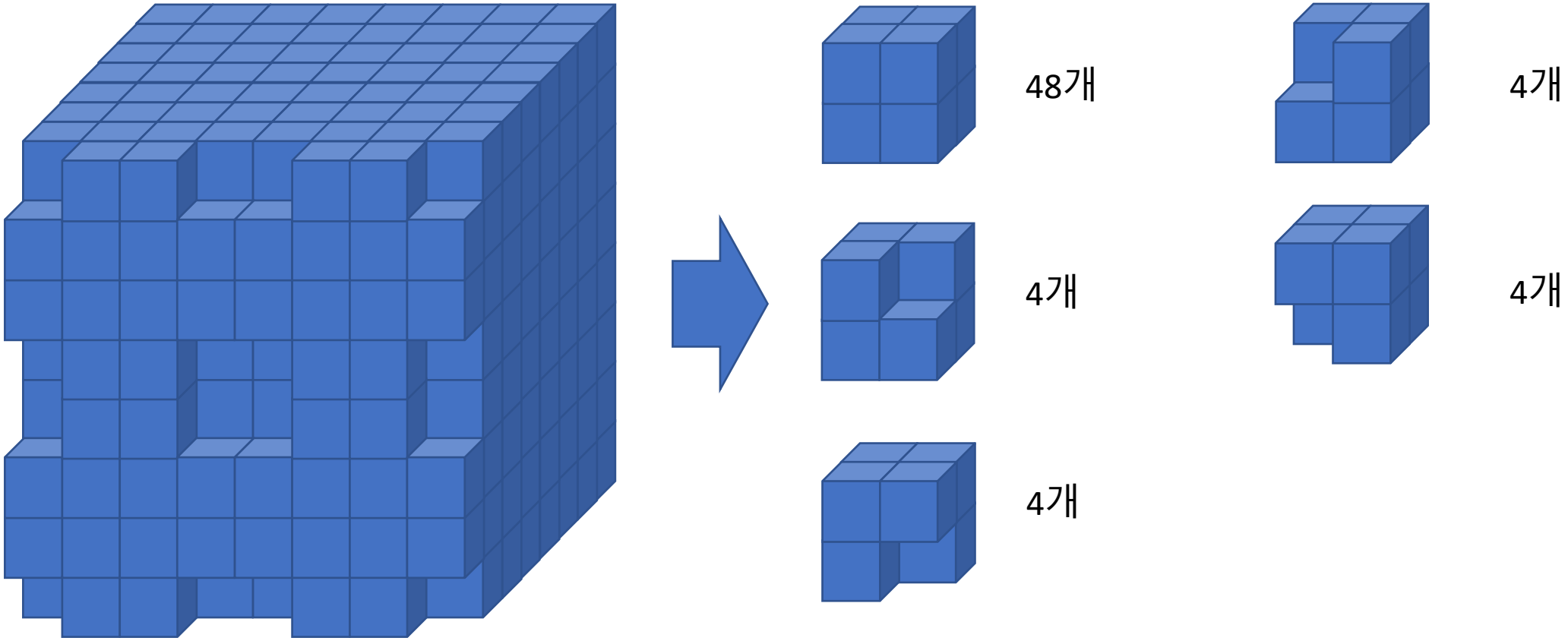
복셀 기하구조 특성

- 복셀월드에서는 꽉 채워진 복셀 오브젝트가 많을 확률이 높다. (산/언덕/지층)
- 적어도 대부분의 경우 중심으로 갈수록 밀도가 높다.
- 2x2x2영역이 꽉 차 있을 가능성이 높다.

복셀 기하구조 압축

1. 8x8x8 복셀 오브젝트에서 2x2x2블럭으로 쪼개면 2x2x2블럭의 패턴은 8 bits, 최대 256가지.
2. 최대 패턴 개수 16개로 제한을 건다. 오브젝트 한개당 16개 패턴을 초과할 경우 압축불가로 처리.
3. 8 bits x 16 = 128 bits = 16 bytes, 이것이 패턴 팔레트 사이즈
4. 2x2x2블럭 하나는 16가지중 하나의 패턴을 가지게 되므로 각 블럭은 4 bits로 표현 가능.

5. 8x8x8오브젝트에서 2x2x2블럭의 개수는 64개. 4 bits x 64 = 256 bits = 32 bytes. 이것이 압축된 복셀 데이터 바디 사이즈
6. 패턴 팔레트16 bytes + 바디 32 bytes = 48 bytes.
7. 압축하지 않은 8x8x8복셀오브젝트의 복셀 데이터 사이즈는 64 bytes.
8. 패턴이 16개인 오브젝트는 64bytes -> 48 bytes 로 25%만큼 사이즈를 줄일 수 있음.
9. 패턴이 8개 이하일때는 2x2x2블럭 하나를 3 bits로 표현 가능. 이 경우 패턴 팔레트 사이즈 8 bytes , 바디 사이즈 3 bits x 64 = 192 bits = 24 bytes , 합쳐서 32 bytes로 압축가능.
10. 패턴이 4개 이하일때 2x2x2블럭 하나를 2 bits로 표현 가능. 패턴 팔레트 사이즈 4 bytes , 바디 사이즈 2 bits x 64 = 128 bits = 16 bytes , 합쳐서 20 bytes로 압축가능.



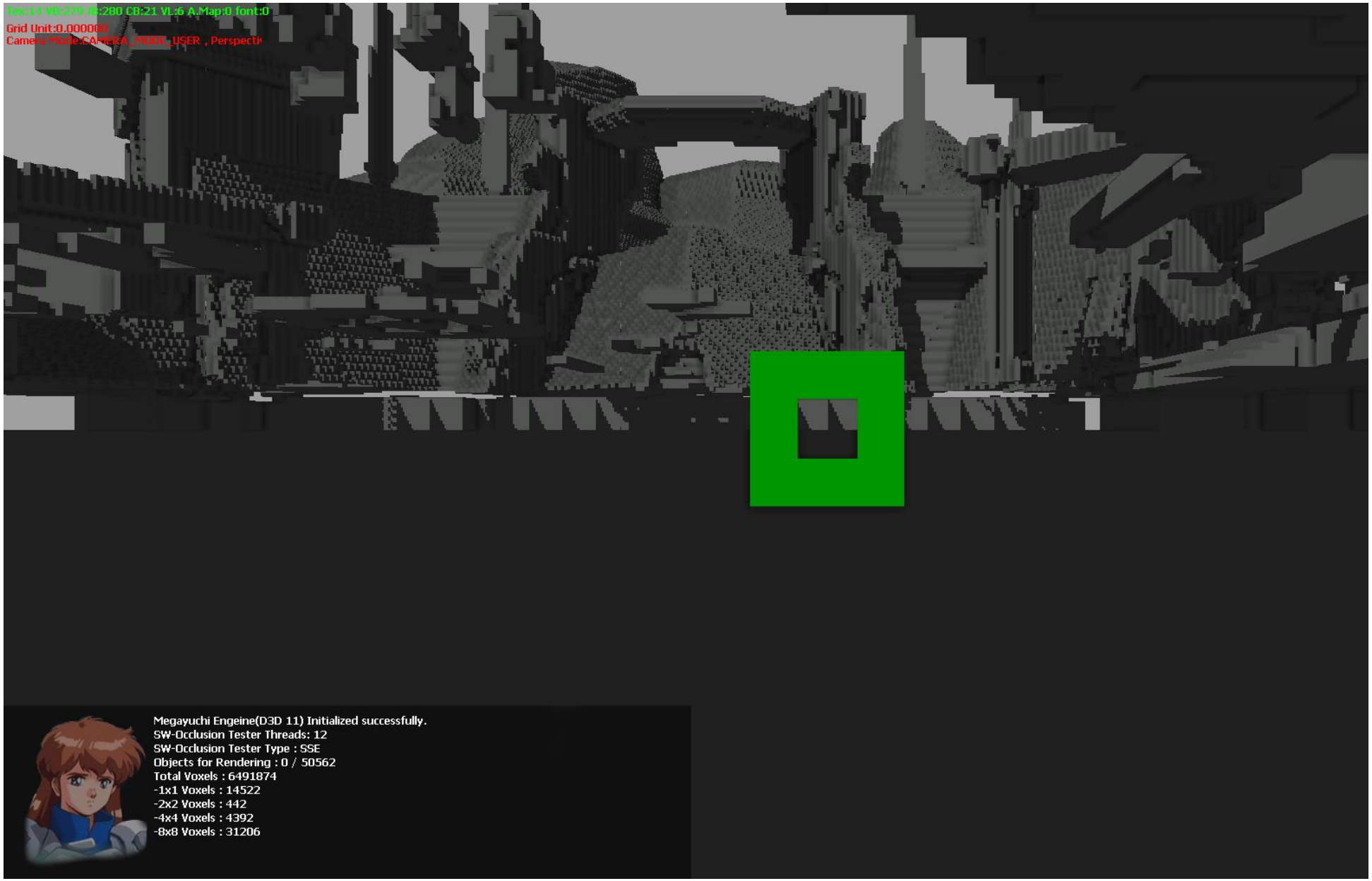
패턴 5개, 8개 미만이므로 패턴 인덱스는 3 Bits
3 Bits x 64 = 192 Bits = 24 Bytes
8x8x8복셀 오브젝트의 기본 사이즈는 512 Bits = 64 Bytes
24 / 64 = 37.5% -> 37.5%로 사이즈 감소

패턴 개수	오브젝트 개수
3	8417
4	8682
5	2370
6	3498
7	1902
8	1686
9	1481
10	1175
11	586
12	402
13	272
14	205
15	153
16	111

총 오브젝트 개수 : 50562

압축 가능한 오브젝트 수 : 30941

8x8x8오브젝트 개수 : 31206



스트리밍시 평균적으로 61%정도의 오브젝트에 대해 25% 패킷사이즈 감소.

복셀 컬러 데이터 특성

- 인접한 복셀은 같거나 비슷한 색깔을 가질 확률이 높다.
- 일반적으로 ‘비슷한 ' 색상은 컬러 팔레트에서의 거리가 가까 운 색상이다.

복셀 컬러 데이터 압축

1. 1x1x1오브젝트는 압축대상에서 제외한다.
2. 복셀 기하구조를 압축할때와 마찬가지로 2x2x2 복셀영역을 한 블럭으로 잡는다.
3. 블럭 내 8개 복셀의 컬러값 중 최소 컬러값을 구한다.
4. 각 블럭에서 구한 Min. Color값을 header에 저장한다. 4x4x4복셀 오브젝트의 경우 2x2x2개의 Min Color 개수가 나온다. 따라서 8bytes를 소모한다.

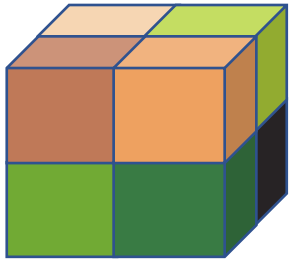
복셀 컬러 테이블 압축

6. 각 복셀의 컬러 인덱스 대신, [해당 색상 인덱스 – Min. Color값]을 body에 저장한다. 최대 127 offset만 허용하므로 복셀당 컬러 데이터는 최대 7bits만 소모한다.
7. 8개 복셀의 컬러가 모두 같을 경우 복셀의 비트값 필드는 -1로 설정

복셀 한칸당 필요한 bit수

```
offset == 0 , bits : 0
offset == 1 , bits : 1
osffet == 2 ~ 3, bits : 2
offset == 4 ~ 7 , bits : 3
offset == 8 ~ 15 , bits : 4
offset == 8 ~ 31 , bits : 5
offset == 8 ~ 63 , bits : 6
offset == 64 ~ 127 , bits : 7
```

2x2x2 복셀 오브젝트의 컬러테이블 압축



top

6	7
4	5

2	3
0	1

bottom

10	11
8	9

6	7
4	5

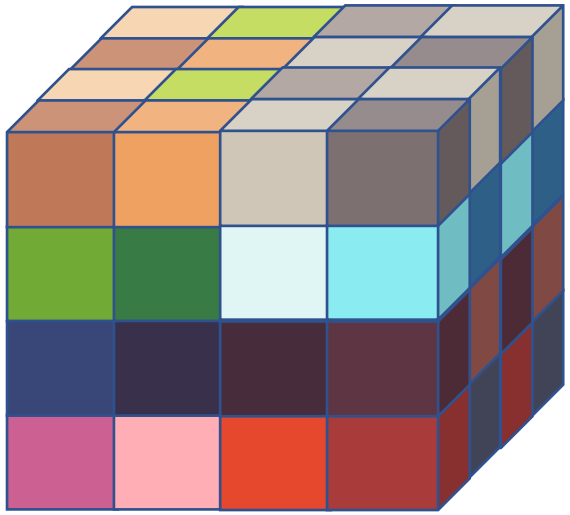
Min. Color = 4 -> 2x2x2블럭의 대표컬러
Max. Distance = 7 -> 7을 표현하기 위한 bits수 = 3 bits -> 복셀당 3bits 필요

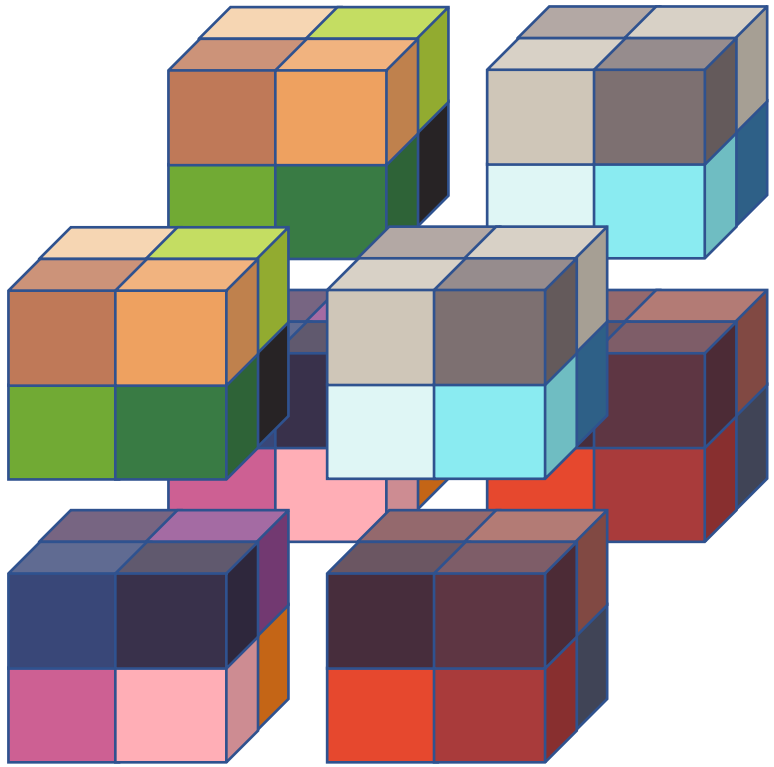
Resolution은 0-127까지지만 2^n으로 표현하면 3bits,
계산의 용이함을 위해 4bits로 표현

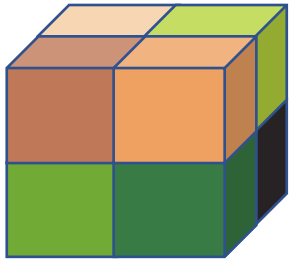
Header
Resolution : 1 byte (2x2x2당 1개의 resolution(4bits) – 1byte로 표현)
Color: 1byte

Body
Data : 3x8 = 24 bits = 3 bytes
1 + 1+ 3 = 5 bytes

4x4x4 복셀 오브젝트의 컬러테이블 압축







top

6	7
4	5

2	3
0	1

bottom

10	11
8	9

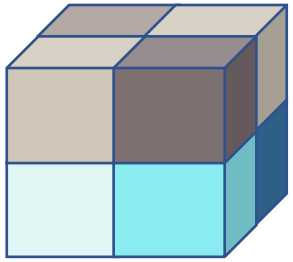
6	7
4	5

Min. Color = 4 -> 2x2x2블럭의 대표컬러
Max. Distance = 7 -> 7을 표현하기 위한 bits수 = 3 bits -> 복셀당 3bits 필요

Resolution은 0-127까지지만 2^n으로 표현하면 3bits,
계산의 용이함을 위해 4bits로 표현

Header
Resolution : 1 byte (2x2x2당 1개의 resolution(4bits) – 1byte로 표현)
Color: 1byte

Body
Data : 3x8 = 24 bits = 3 bytes
1 + 1+ 3 = 5 bytes



top

14	15
12	13

2	3
0	1

bottom

18	19
16	17

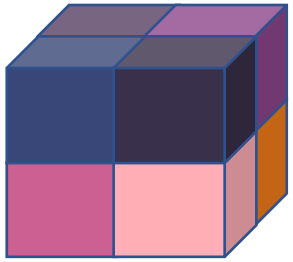
6	7
4	5

Min. Color = 4 -> 2x2x2블럭의 대표컬러
Max. Distance = 7 -> 7을 표현하기 위한 bits수 = 3 bits -> 복셀당 3bits 필요

Resolution은 0-127까지지만 2^n으로 표현하면 3bits,
계산의 용이함을 위해 4bits로 표현

Header
Resolution : 1 byte (2x2x2당 1개의 resolution(4bits) – 1byte로 표현)
Color: 1byte

Body
Data : 3x8 = 24 bits = 3 bytes
1 + 1+ 3 = 5 bytes



top

22	23
20	21

2	3
0	1

bottom

26	27
24	25

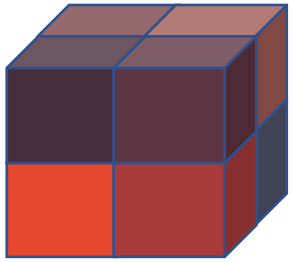
6	7
4	5

Min. Color = 4 -> 2x2x2블럭의 대표컬러
Max. Distance = 7 -> 7을 표현하기 위한 bits수 = 3 bits -> 복셀당 3bits 필요

Resolution은 0-127까지지만 2^n으로 표현하면 3bits,
계산의 용이함을 위해 4bits로 표현

Header
Resolution : 1 byte (2x2x2당 1개의 resolution(4bits) – 1byte로 표현)
Color: 1byte

Body
Data : 3x8 = 24 bits = 3 bytes
1 + 1+ 3 = 5 bytes



top

2	3
0	1

2	3
0	1

bottom

30	31
28	29

30	31
28	29

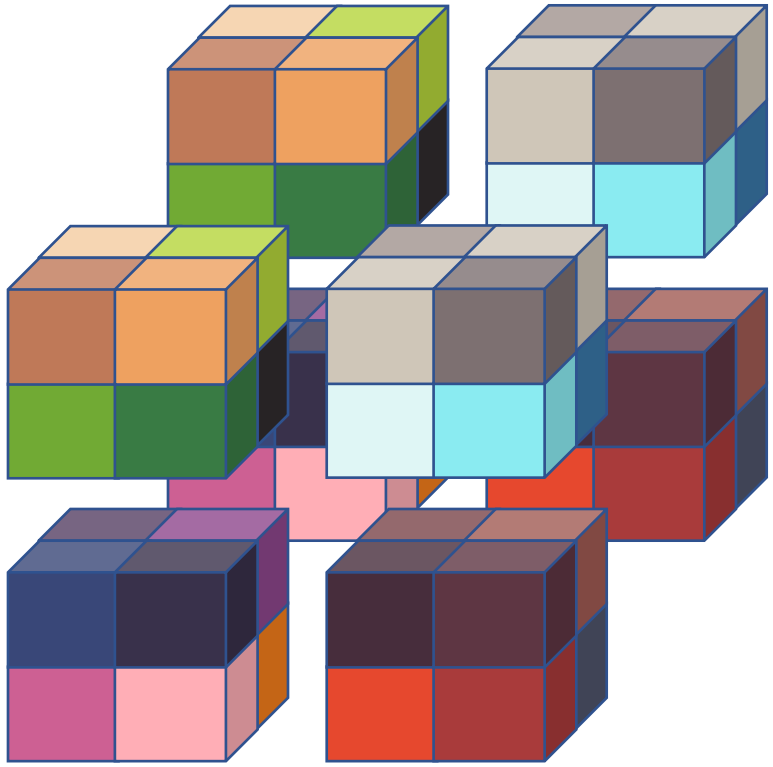
Min. Color = 0 -> 2x2x2블럭의 대표컬러
Max. Distance = 31 -> 31을 표현하기 위한 bits수 = 5 bits -> 복셀당 5 bits 필요

Resolution은 0-127까지지만 2^n으로 표현하면 3bits,
계산의 용이함을 위해 4bits로 표현

Header
Resolution : 1 byte (2x2x2당 1개의 resolution(4bits) – 1byte로 표현)
Color: 1byte

Body
Data : 5x8 = 40 bits = 5 bytes
1 + 1+ 5 = 7 bytes

4x4x4 복셀 오브젝트의 압축



Min. Color = 4
Max. Distance = 7, 복셀당 3bits 필요

Min. Color = 12 -> 2x2x2블럭의 대표컬러
Max. Distance = 7, 복셀당 3bits 필요

Min. Color = 20 -> 2x2x2블럭의 대표컬러
Max. Distance = 7, 복셀당 3bits 필요

Min. Color = 0 -> 2x2x2블럭의 대표컬러
Max. Distance = 31, 복셀당 5 bits 필요

Min. Color = 4 -> 2x2x2블럭의 대표컬러
Max. Distance = 7, 복셀당 3bits 필요

Min. Color = 12 -> 2x2x2블럭의 대표컬러
Max. Distance = 7, 복셀당 3bits 필요

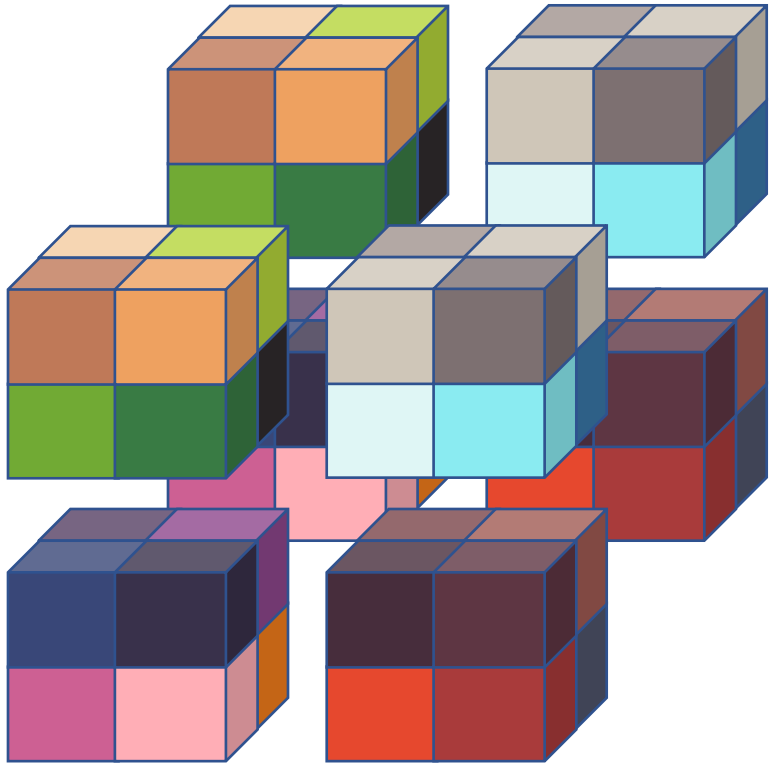
Min. Color = 20 -> 2x2x2블럭의 대표컬러
Max. Distance = 7, 복셀당 3bits 필요

Min. Color = 0 -> 2x2x2블럭의 대표컬러
Max. Distance = 31, 복셀당 5 bits 필요

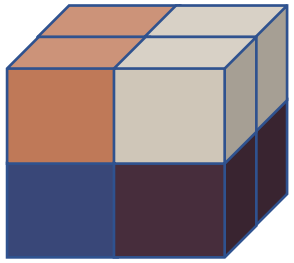
Header
Resolution : 4 byte (2x2x2당 1개의 resolution(4bits) x 2x2x2 = 32 bits)
Min. Color Table: 1 x 2x2x2 = 8byte

Body
Data : 8x3 + 8x3 + 8x3 + 8x5 + 8x3 + 8x3 + 8x3 + 8x5 = 28 bytes
4 + 8 + 28 = 40 bytes

Min. Color 테이블 압축



Min. Color Table



압축가능!

Min. Color = 4
Max. Distance = 7, 복셀당 3bits 필요

Min. Color = 12 -> 2x2x2블럭의 대표컬러
Max. Distance = 7, 복셀당 3bits 필요

Min. Color = 20 -> 2x2x2블럭의 대표컬러
Max. Distance = 7, 복셀당 3bits 필요

Min. Color = 0 -> 2x2x2블럭의 대표컬러
Max. Distance = 31, 복셀당 5 bits 필요

Min. Color = 4 -> 2x2x2블럭의 대표컬러
Max. Distance = 7, 복셀당 3bits 필요

Min. Color = 12 -> 2x2x2블럭의 대표컬러
Max. Distance = 7, 복셀당 3bits 필요

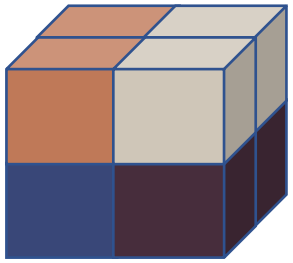
Min. Color = 20 -> 2x2x2블럭의 대표컬러
Max. Distance = 7, 복셀당 3bits 필요

Min. Color = 0 -> 2x2x2블럭의 대표컬러
Max. Distance = 31, 복셀당 5 bits 필요

Header
Resolution : 4 byte (2x2x2당 1개의 resolution(4bits) x 2x2x2 = 32 bits)
Min. Color Table: 1 x 2x2x2 = 8byte

Body
Data : 8x3 + 8x3 + 8x3 + 8x5 + 8x3 + 8x3 + 8x3 + 8x5 = 28 bytes
4 + 8 + 28 = 40 bytes

Min. Color 테이블 압축



top

4	12
4	12

4	12
4	12

bottom

20	0
20	0

20	0
20	0

Min. Color = 0
Max. Distance = 20 -> 20을 표현하기 위한 bits수 = 5 bits -> 복셀당 5bits 필요

Header
Resolution : 1 byte
Color: 1byte

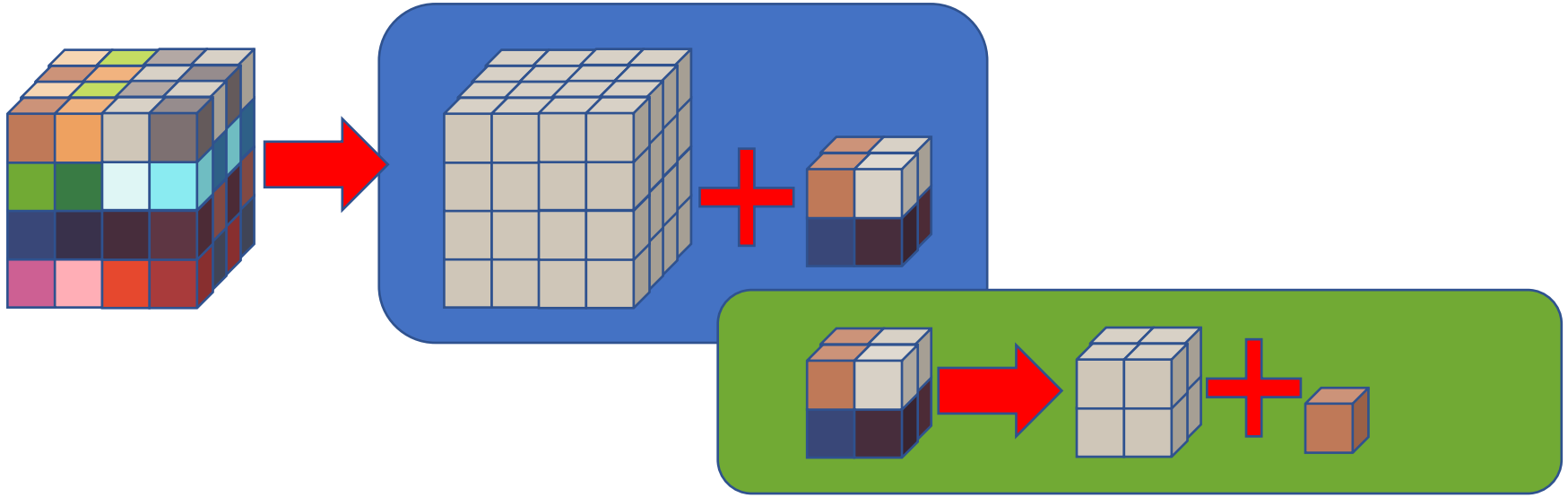
Body
Data : 5x8 = 40 bits = 5bytes
1 + 1 + 5 = 7 bytes

Min. Color 테이블의 재귀적 압축

2x2x2 오브젝트의 컬러테이블을 압축하면
{
  1x1x1x Min. Color 테이블
  2x2x2 N bits distance 테이블
}

Min. Color 테이블의 재귀적 압축

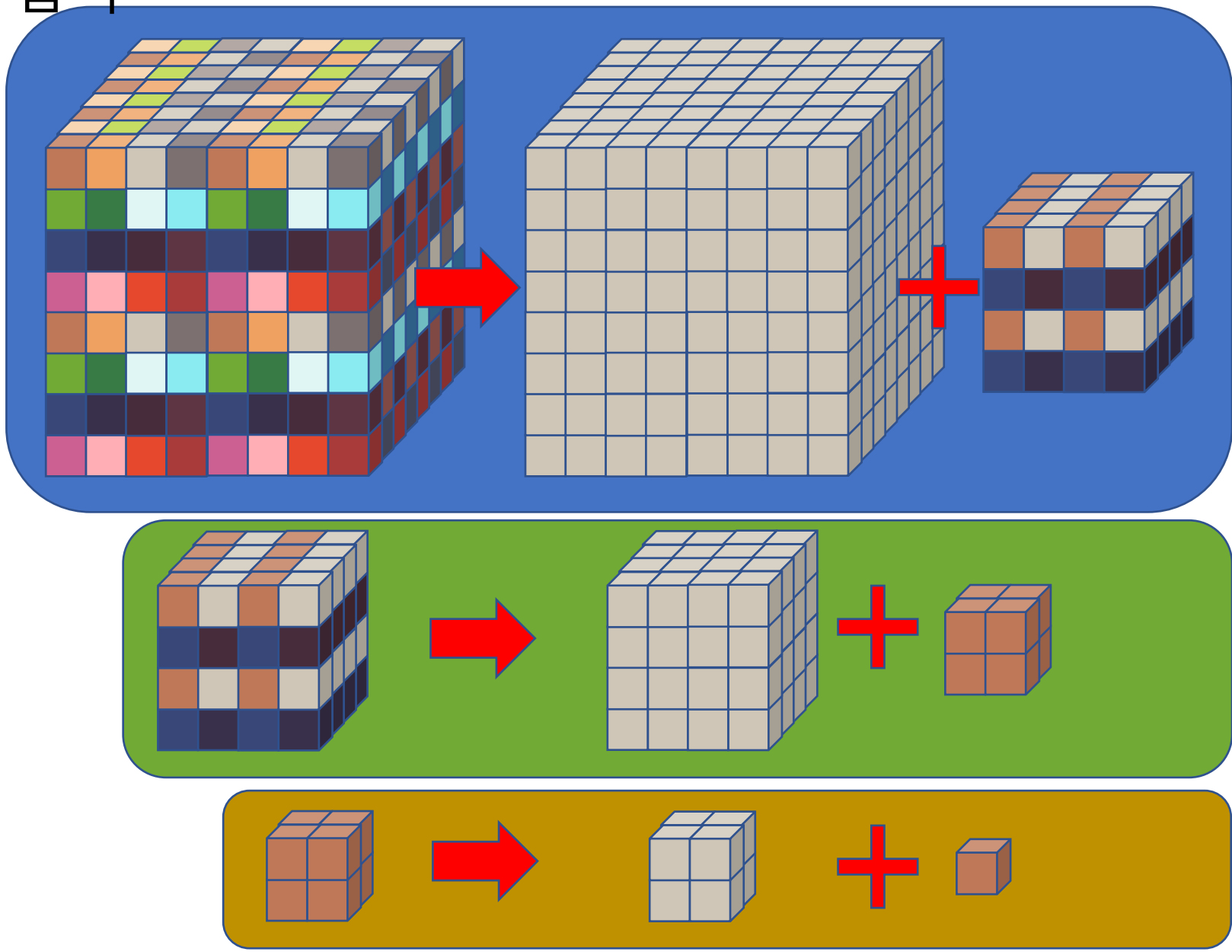
4x4x4 오브젝트의 컬러테이블을 압축하면
{
  2x2x2 Min, Color 테이블(압축가능)
  {
    1x1x1x Min. Color 테이블
    2x2x2 N bits distance 테이블
  }
  4x4x4 N bits distance 테이블
}



Min. Color 테이블의 재귀적 압축

8x8x8 오브젝트의 컬러테이블을 압축하면

{
  4x4x4 Min, Color 테이블(압축가능)
  {
    2x2x2 Min. Color 테이블(압축가능)
    {
      1x1x1 Min. Color 테이블
      2x2x2 N bits distance 테이블
    }
    4x4x4 N bits distance 테이블
  }
  8x8x8 N bits distance 테이블
}



압축 테스트

- 클라이언트에서 save_voxels_packet xxx.vxp
- 클라이언트에서 compress
- 서버에서 메모리 리포트 – F7

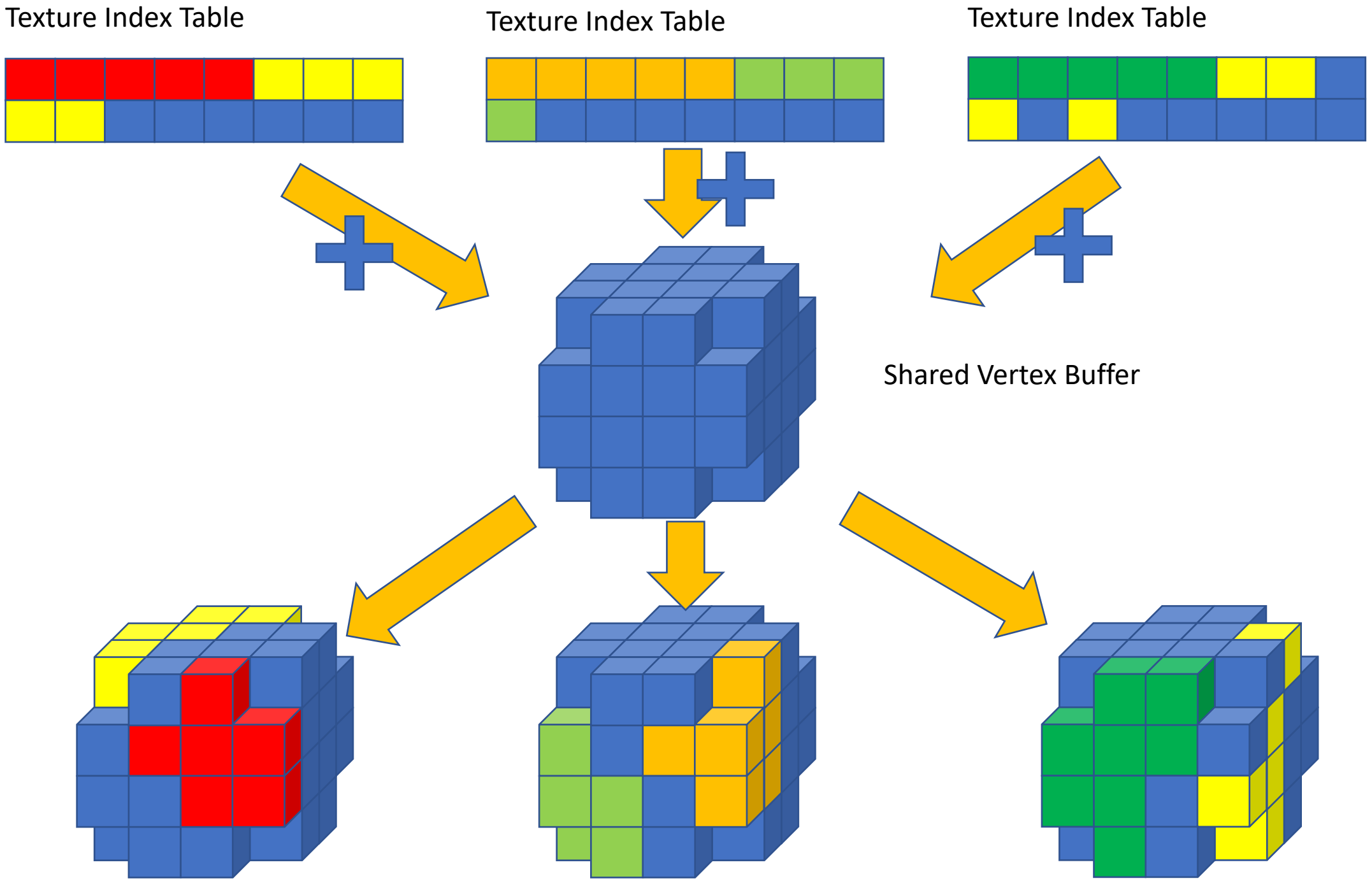
압축 타이밍(서버에서)

- 기하구조나 컬러테이블에 변화가 생기면 압축처리 큐에 추가
- 주기적으로 큐에 쌓여있는 오브젝트들에 대해서 멀티스레드로 데이터 압축 처리.
- 멀티스레드 + 지연된 처리
- 서버 -> 클라이언트로 전송해야할 이벤트가 발생했는데 해당 오브젝트의 압축작업이 완료되지 않았다면 그 즉시 압축처리.

기타 최적화

버텍스 버퍼 Sharing

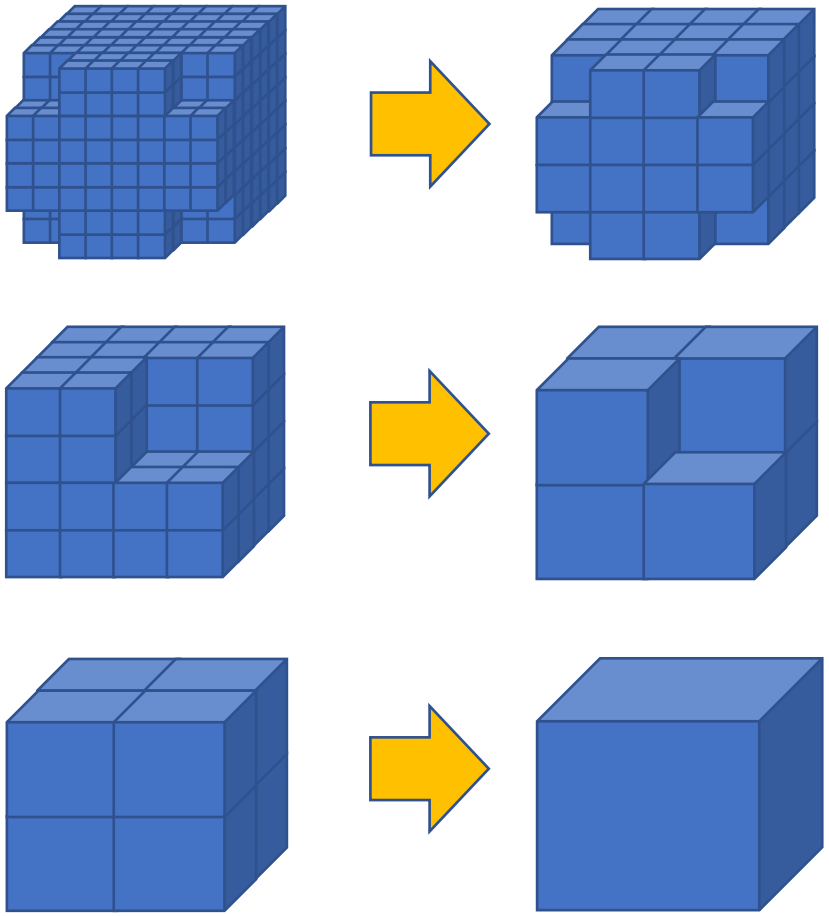
- 8x8x8짜리 복셀 조합은 어마어마하게 많은 경우의 수를 만들어낸다.
- 하지만 월드를 구성해보면 중복되는 패턴도 많이 나온다.
- 컬러(텍스처) 조합을 고려하면 훨씬 많은 조합이 있지만 이미 텍스처 인덱스 테이블을 분리했다!
- 따라서 8x8x8 복셀 조합은 중복되는 패턴들에 대해서 한번 만들어 놓은 VertexBuffer는 그대로 재활용 가능.
- 상당한 GPU메모리 절약 효과가 있다.


Texture Index Table
Texture Index Table
Texture Index Table
Shared Vertex Buffer

성능과 메모리 절약을 위한 복셀 최적화

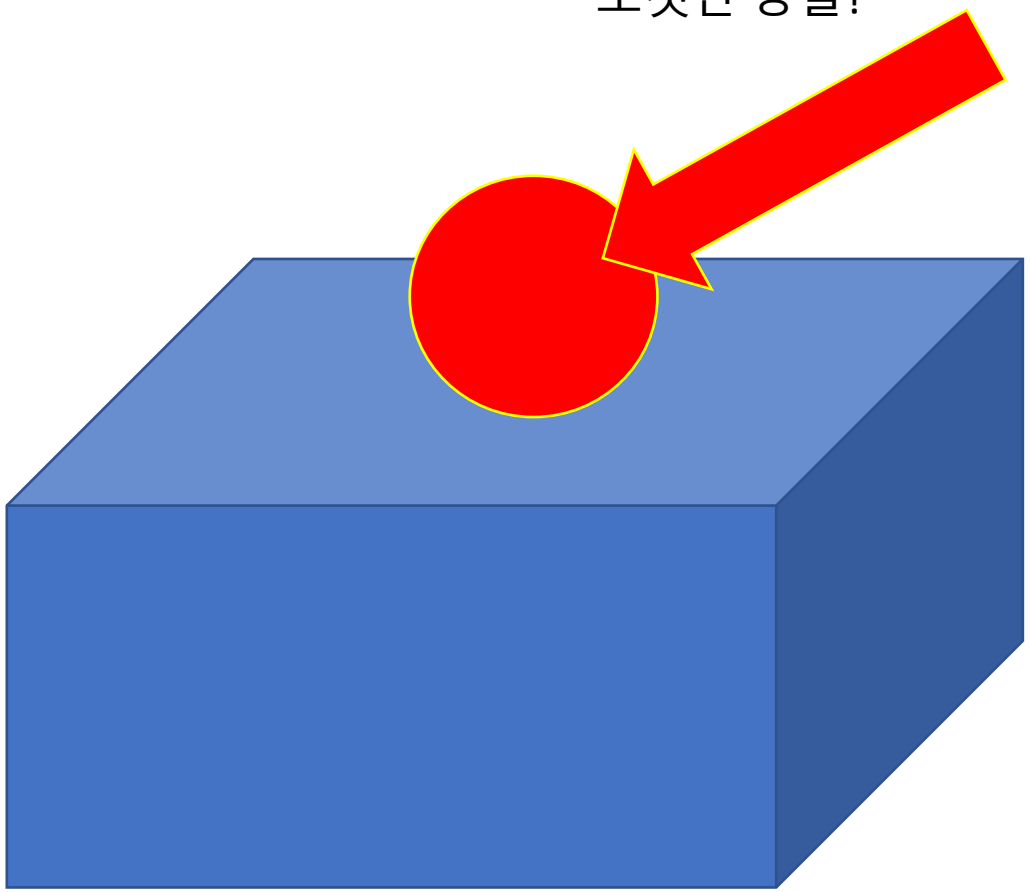
- 복셀 데이터의 정밀도는 가능한 낮게 유지한다.
- 서버 스타트 후 복셀 데이터 로드 후 복셀 데이터를 최적화.
- 모양이 변형되지 않는 선에서 최대한 낮은 정밀도로 변환.
- 50cm 단위로 복셀을 편집하거나 로켓탄이 터져서 8x8x8정밀도가 필요한 상황이 되면 즉시 8x8x8정밀도로 변환.
- 변환후에도 복셀의 기하구조를 직접 전송하지 않음. 변환 룰은 명백하므로 클라이언트에서도 알아서 변환.

모양을 유지하는 선에서 최적화



모양을 유지하는 선에서 최적화

로켓탄 충돌!



최대한 낮은 정밀도로 유지

8x8x8로 변환