Voxel Rendering

간단소개

김성익 (http://gamecode.org)

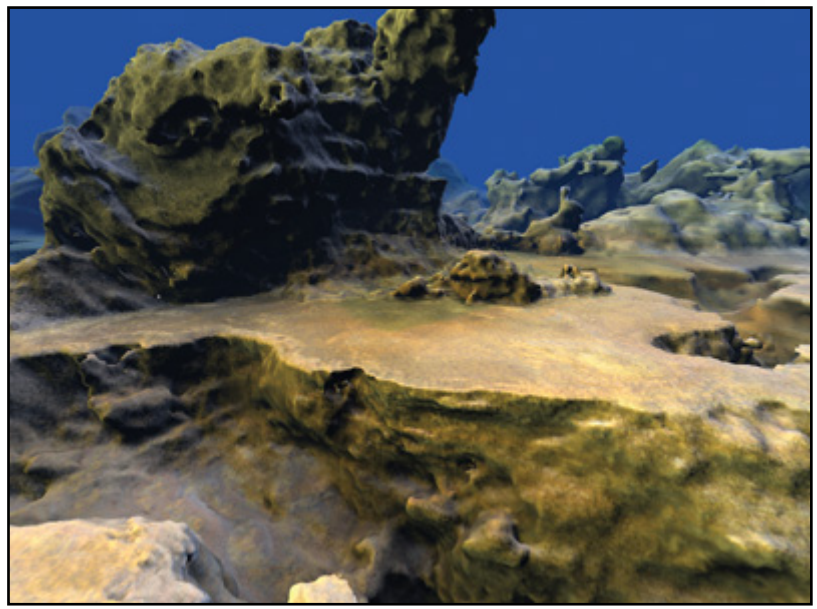
2009/10/25

내용

- Voxel 렌더링 소개
  - Voxel Field 렌더링
    - Marching Cube 를 이용한 ISO Surface 렌더링
  - Voxel Volume 렌더링
    - Ray Casting
    - Sparse Voxel Octrees

Voxel Field 렌더링

- Voxel 데이터
  - 3차원 Array
  - Field 값
- 렌더링
  - 값이 일정한 표면을 렌더링

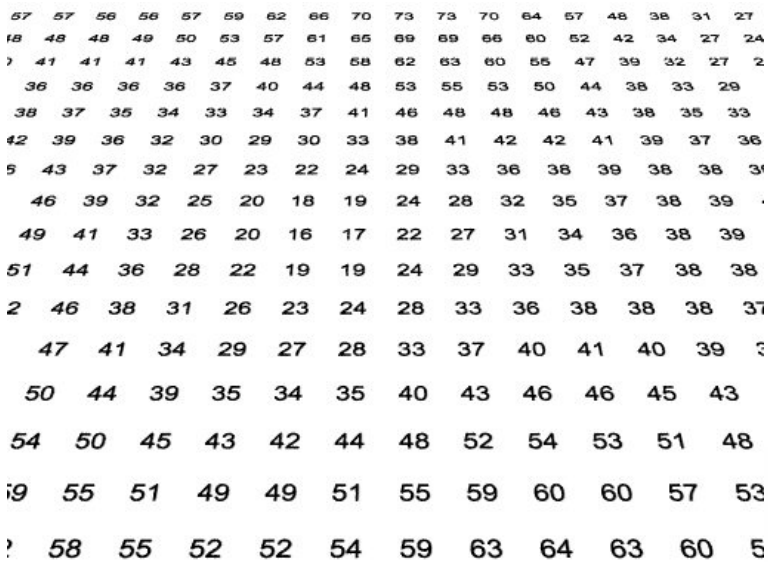


http://www.geisswerks.com/about_terrain.html

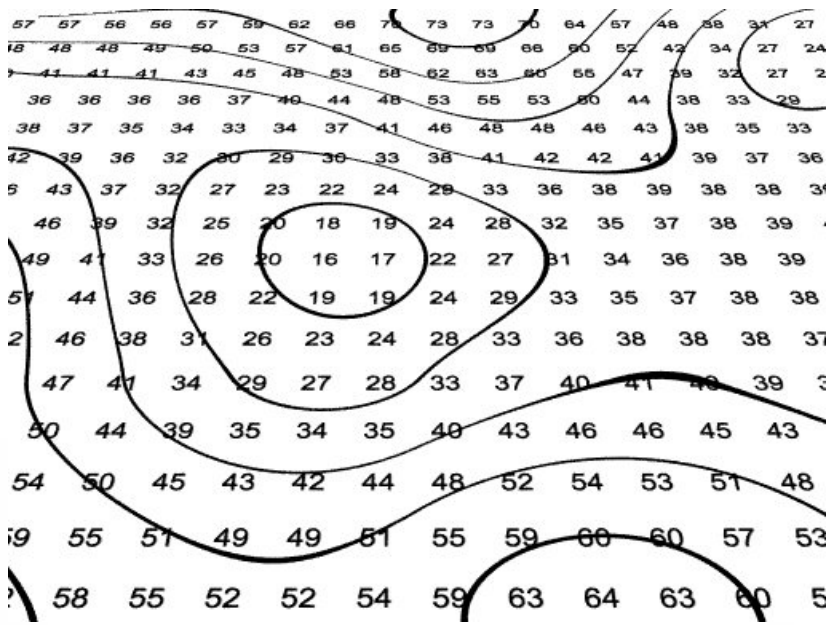
- ISO Bar
  - Marching Square Algorithm
- ISO Surface
  - Marching Cube Algorithm

ISO Bar (1)

- 데이터 : 2d 그리드에 값
- 값이 일정한 지역을 라인으로 렌더링 (등고선)

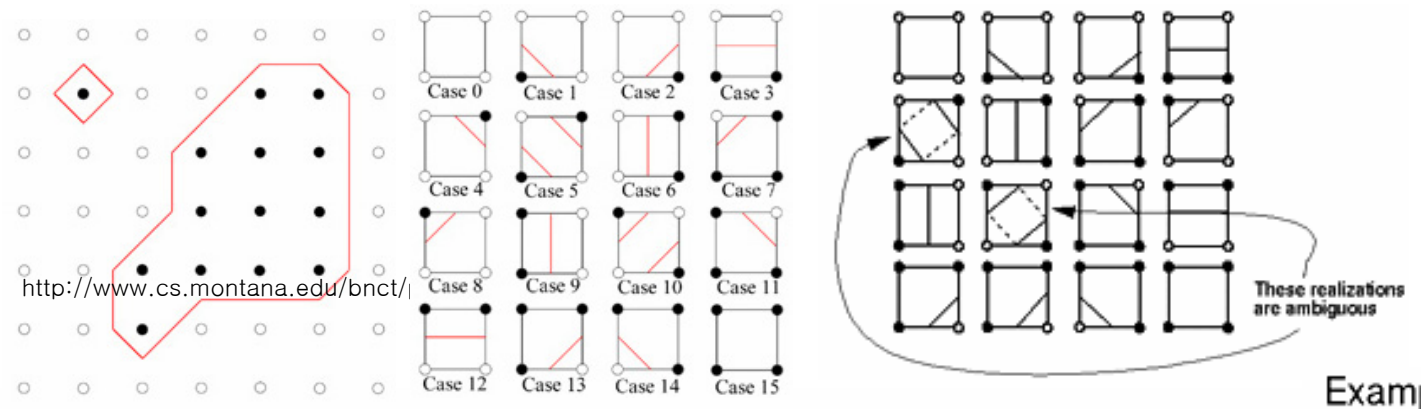


http://www.econym.demon.co.uk/isotut/isobars.htm

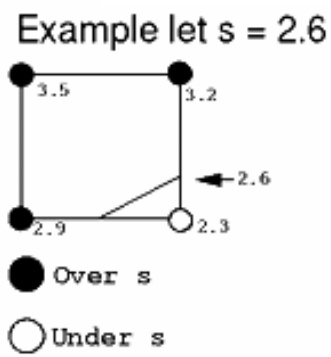


ISO Bar (2)

- Marching Square
  - 인접한 4개의 점으로 이루어진 유닛 단위
  - 표면값보다 크거나 같은지, 혹은 작은지에 따라 라인이 생기는 패턴이 결정
  - 패턴이 대칭 및 회전하는 특성 (패턴의 수를 줄일 수 있다)



  - 값의 비율에 따라 라인 위의 버텍스 위치 결정

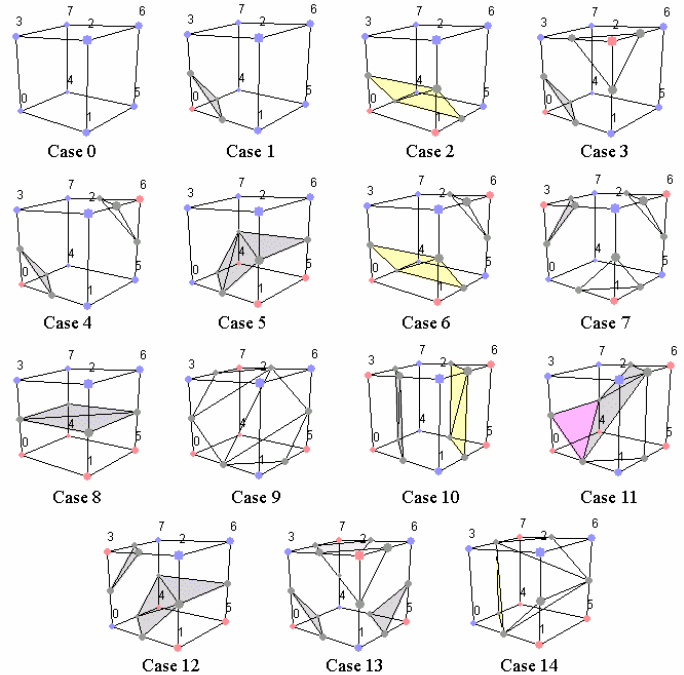


ISO Surface(1)

- ISO-Bar 의 확장형
- 데이터 : 3차원 필드 데이터
- 값이 일정한 표면을 렌더링

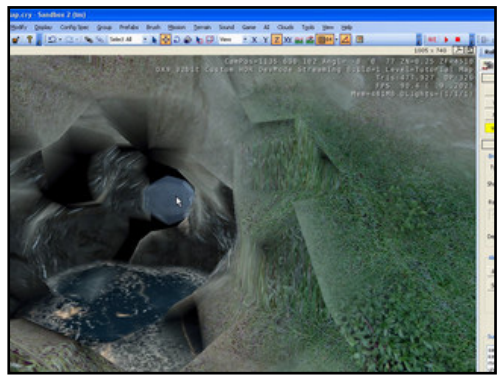
- Marching Cube 알고리즘
  - 인접한 버텍스를 공유하는 유닛
  - 256가지 타입의 패턴
  - 패턴화 (14가지 고유 패턴)

- 3D Voxel Field => 표면 Polygon 화
  - 전처리 가능 (로딩 타임)



ISO Surface(2)

- 적용 사례
  - Crysis Voxel Terrain
  - Blob
  - GPU Gems 3
    Generating Complex Procedural Terrains Using the GPU



- 특징
  - 3D Voxel 데이터를 손쉽게 표현 가능
    - 동굴 같은 지형 표현 가능
    - 손쉬운 터레인 편집 (3d 어레이 값만 간단하게 에디팅)
  - DCC툴 부재로 인한 인하우스 에디터 필요
  - 텍스처 매핑의 어려움
    - Nvidia Case : UV 는 normal 활용, Splatting 기법 활용
    - 각 유닛 별 별도의 UV 채널을 설정하는 방법도 가능

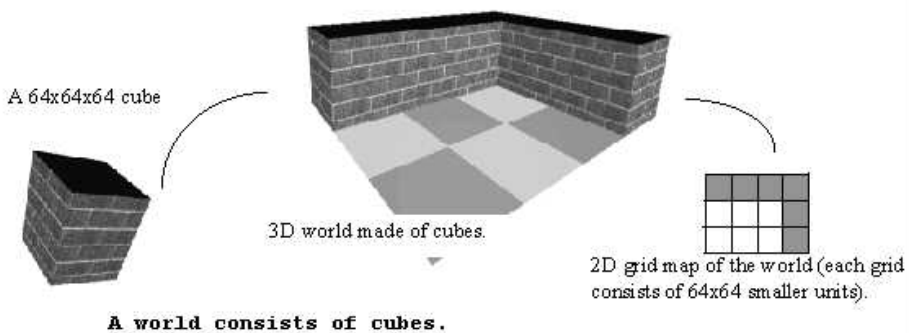
Voxel Volume Rendering

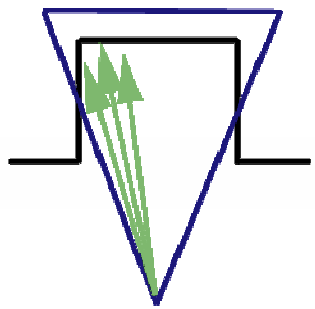
- 데이터
  - 복셀
  - 3차원 컬러 데이터
- 렌더링
  - 화면에 복셀 이미지를 렌더링
- 차세대 렌더링 기법

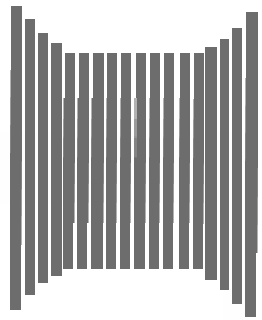
- Ray Casting
  - Classic Game Ray casting
  - Volume Ray casting
- Octree
- Sparse Voxel Octree

RayCasting

- 울펜스타인 3d, Doom 시절의 Ray Casting
  - 바닥 + 벽 + 천정으로 분리
  - 벽 부분을 Raycasting
  - 데이터 : 벽 2d 맵
    - 시야 방향으로 데이터 탐색해서 벽 정보 계산
    - 맵 기반으로 충돌 지점 계산
    - 가로 라인에 대해서만 계산 (속도가 빠름)
  - 적은 2d 정보를 활용해서 월드의 모든 벽 정보 표현

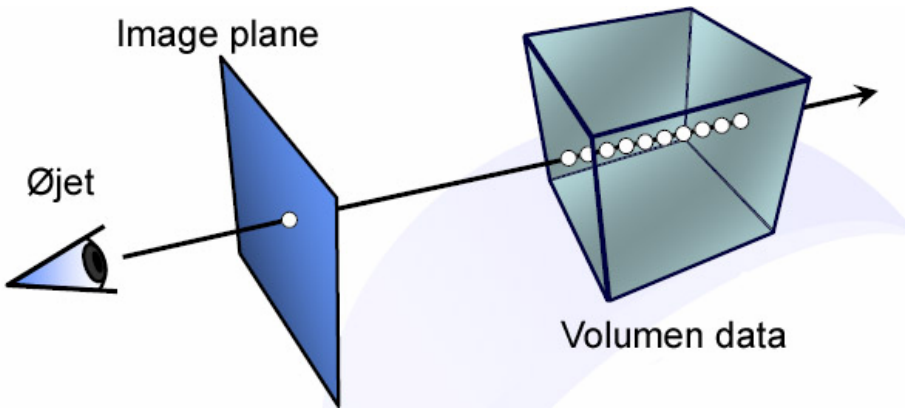




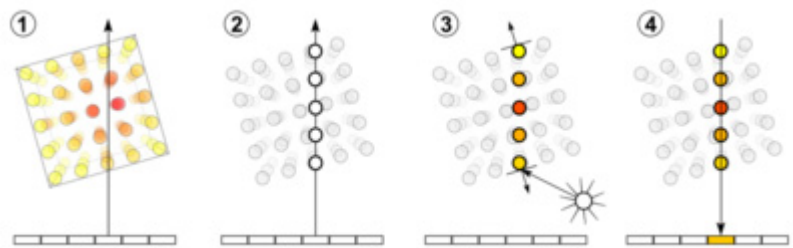


Volume Raycasting (1)

- 3D Voxel 데이터를 검색
- 뷰에서 Ray 를 쏘아서 Voxel 데이터 상의 충돌 지점을 검출하여 해당 픽셀을 렌더링



- 많은 셈플링이 필요함



Volume Raycasting(2)

- 적용예 (http://www.daimi.au.dk/~trier/?page_id=98)
  - 많은 샘플링이 필요함

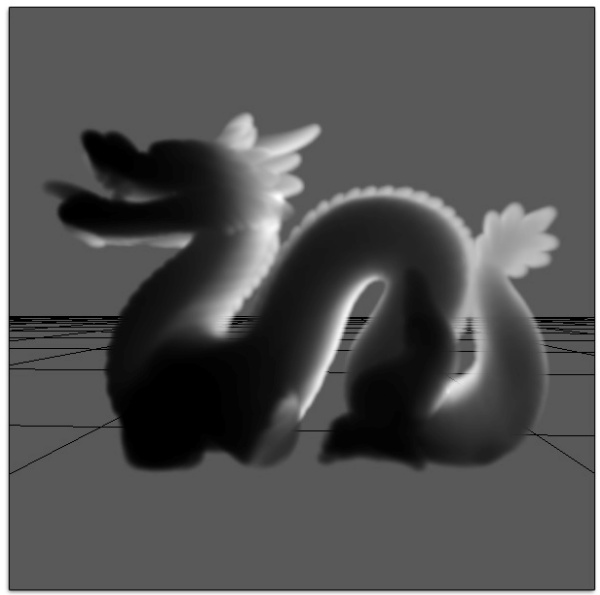
```
fragment_out OUT;
float2 texc = ((IN.Pos.xy / IN.Pos.w) + 1) / 2; // find the right place to lookup in the backside buffer
float4 start = IN.TexCoord; // the start position of the ray is stored in the texturecoordinate
float4 back_position  = tex2D(tex, texc);
float3 dir = float3(0,0,0);
dir.x = back_position.x - start.x;
dir.y = back_position.y - start.y;
dir.z = back_position.z - start.z;
float len = length(dir.xyz); // the length from front to back is calculated and used to terminate the ray
float3 norm_dir = normalize(dir);
float delta = stepsize;
float3 delta_dir = norm_dir * delta;
float delta_dir_len = length(delta_dir);
float3 vec = start;
float4 col_acc = float4(0,0,0,0);
float alpha_acc = 0;
float length_acc = 0;
float4 color_sample;
float alpha_sample;

for(int i = 0; i < 450; i++)
{
  color_sample = tex3D(volume_tex,vec);
  if (color_sample.a < 0.1)
  {
    col_acc = float4(color_sample.rgb, 1);
    break;
  }
  vec += delta_dir;
  length_acc += delta_dir_len;
  if(length_acc >= len)
     break;
}

OUT.Color =  col_acc;
```

Volume Raycasting (3)

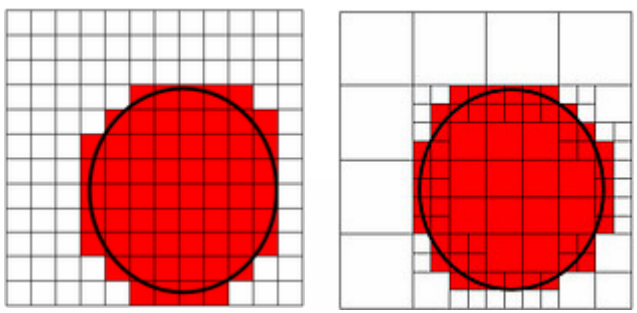
- 특징
  - 복잡한 이미지를 간단한 알고리즘으로 구현할 수 있다
  - 빛의 방향에서의 레이케스팅을 통해서 그림자를 판단할 수 있다
  - 투명한 표면처리가 가능하다
    - 표면의 두께를 판단할 수 있다
    - 진짜 Subsurface Scattering
  - 차세대 렌더링 퀄리티 !!!
  - 용량이 크다
    - 1024x1024x1024 크기의 RGBA 데이터는 무려 4 Giga Byte
  - 연산량이 많다



Voxel Octree

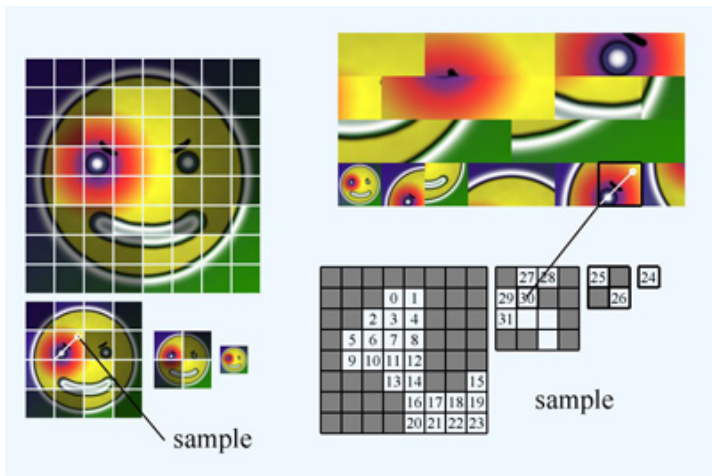
- Voxel 3d 이미지의 치명적인 단점 : 용량
- 용량이 크다 = 샘플링을 많이 해야 한다 = 많은 연산이 필요하다

- Octree 를 활용
  - 용량을 줄인다 => 연산량을 줄인다
    - 속도를 높일 수 있다
  - 단, GPU로 구현 난이도 상승



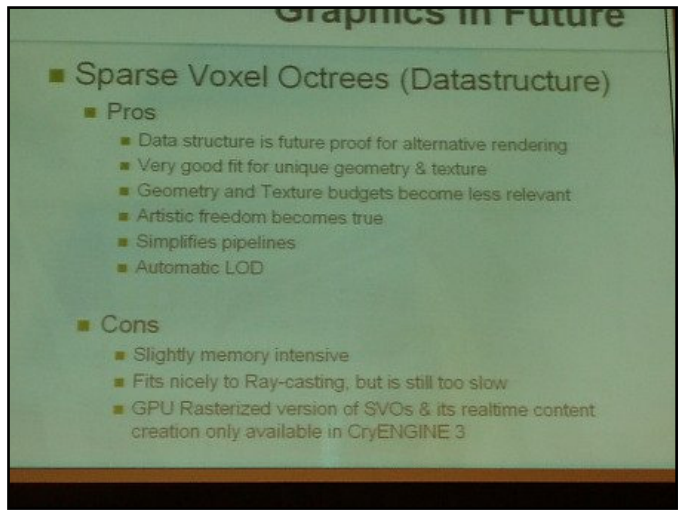
Sparse Virtual Texture

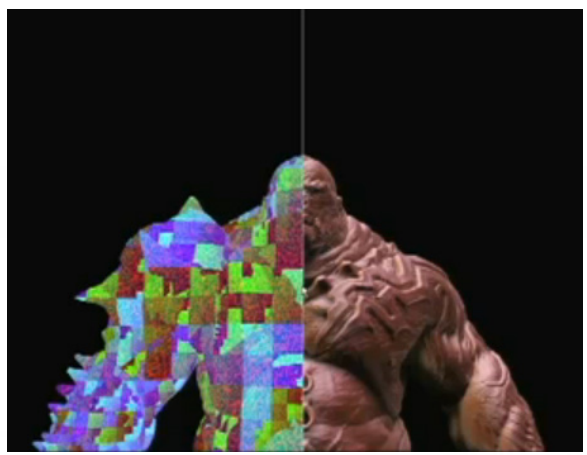
- Sparse Virtual Texture (aka MEGA texture)
  - 운영체제의 가상 메모리 운영하는 것과 비슷
    - 인덱스화 테이블화
  - 철저하게 화면에 맞는 디테일의 데이터만 적제 (블럭단위 LOD)
    - 텍스처 사용량 최소화
    - 비주얼 손실 굉장히 적음
    - 속도 느리지 않음
      - 텍스처 메모리 활용도 증가로 오히려 빠를 수 있음
  - 스트리밍화 (초대형 텍스처 사용 가능)
  - 구현의 복잡함
    - 테이블화
    - 복잡한 텍스처 매니징 시스템
      - 거의 OS 가상 메모리 매니징급 처리
  - Crysis, Unreal3에 적용되어 있음



Sparse Voxel Octree (1)

- 차세대 렌더링 기법으로 지목
  - 크라이텍 GDC 2009 유럽, KGC 2009 키노트
- Voxel 도 SVT와 유사하게 테이블화
  - 텍스처 MipMap 개념과 유사
  - Octree 구조
  - 가상 메모리와 유사한 페이지 테이블
  - 스트리밍 로딩
  - 픽셀 압축
- **SVO (aka Giga Voxel)**
  - Jon Olink (id soft)
  - http://www.youtube.com/watch?v=VpEpAFGplnI





Sparse Voxel Octrees(2)

- 특징
  - 3D 복셀 이미지 렌더링
  - 화면에 보일 최적의 메모리 사용
  - 적은 메모리 = 적은 셈플링 = 적은 연산
  - 복잡한 픽셀 셰이더 연산
    - But, 현 세대 하드웨어도 렌더링 가능
  - 적용 가능한 Asset제작을 위한 DCC툴 부재
    - 인하우스 툴 개발 필요
    - 폴리곤 => Voxel 화
  - 해결 과제 : 레스터라이즈 시스템과의 궁합
    - Depth 버퍼 문제
    - 볼륨 문제
- 참고 : Zbrush (미래의 비전 제시???)
  - 이미 현 시대에서 활용되고 있음
  - 폴리곤 레스터라이징 방식으로는 따라하기 힘든 뷰포트상 모델 렌더링 퀄리티
  - 충격적인 퀄리티의 게임 등장 예상???

ZBRUSH Viewport http://vaelberx.egloos.com/1849742

Q/A